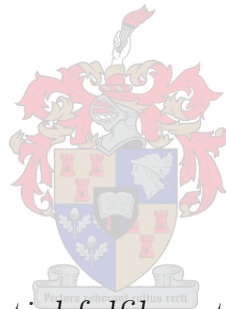


A probabilistic graphical model approach to solving the structure and motion problem

by

Simon Streicher



*Thesis presented in partial fulfilment of the requirements for
the degree of Master of Science in Applied Mathematics in
the Faculty of Science at Stellenbosch University*

Supervisors: Dr Willie Brink and Prof. Johan du Preez

March 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2016

Copyright © 2016 Stellenbosch University
All rights reserved.

Abstract

Probabilistic graphical models show great promise in resolving uncertainty within large systems by using probability theory. However, the focus is usually on problems with a discrete representation, or problems with linear dependencies.

The focus of this study is on graphical models as a means to solve a nonlinear system, specifically the structure and motion problem. For a given system, our proposed solution makes use of multivariate Gaussians to model parameters as random variables, and sigma point linearisation to capture all interrelationships as covariances. This technique does not need in-depth knowledge about given nonlinearities (such as Jacobian matrices) and can therefore be used as part of a general solution.

The aim of structure and motion is to generate a 3D reconstruction of a scene and camera poses, using 2D images as input. We discuss the typical feature based structure and motion pipeline along with the underlying multiview geometry, and use this theory to find relationships between variables.

We test our approach by building a probabilistic graphical model for the structure and motion problem and evaluating it on different types of synthetic datasets. Furthermore, we test our approach on two real-world datasets.

From this study we conclude that, for structure and motion, there is clear promise in the performance of our system, especially on small datasets. The required runtime quickly increases, and the accuracy of results decreases, as the number of feature points and camera poses increase or the noise in the inputs increase. However, we believe that further developments can improve the system to the point where it can be used as a practical and robust solution for a wide range of real-world image sets. We further conclude that this method can be a great aid in solving similar types of nonlinear problems where uncertainty needs to be dealt with, especially those without well-known solutions.

Opsomming

In waarskynlikheidsleer slaag grafiese modelle daarin om onsekerheid in groot stelsels op te los. Die fokus is egter gewoonlik op stelsels met 'n diskrete voorstelling, of met lineêre afhanklikhede.

In hierdie studie fokus ons op grafiese modelle as 'n oplossing vir 'n nie-lineêre probleem, die struktuur-en-bewegingsbepalingprobleem. Ons voorgestelde oplossing maak gebruik van Gaussiese meerveranderlikes om 'n gegewe probleem se parameters in stogastiese veranderlikes te parameteriseer en sigmapunt-linearisering om al die interafhanklikhede as kovariansies voor te stel. Hierdie tegniek benodig geen in-diepte kennis oor die gegewe nie-lineariteite nie (soos bv. die Jacobiaanmatriks), en kan dus gebruik word as deel van 'n algemene oplossing.

Die doel van struktuur-en-bewegingsbepaling is om 'n 3D-struktuur en kamera-posisies te bepaal, met 2D-beelde as intree. Ons bespreek die tipiese pyplyn vir beeldkenmerkgebaseerde struktuur-en-bewegingsbepaling en die onderliggende multivisiemeetkunde wat daarmee gepaard gaan, en gebruik hierdie teorie om die verhoudings tussen veranderlikes voor te stel.

Ons toets ons benadering deur 'n grafiese model van struktuur-en-bewegingsbepaling op te stel en die resultate te evalueer met betrekking tot verskillende tipes sintetiese datastelle. Ons toets ook ons benadering op twee werklike datastelle.

Hierdie studie lei ons tot die gevolgtrekking dat ons sisteem belowende resultate wys vir struktuur-en-bewegingsbepaling. Die uitvoertyd neem vinnig toe, en die akkuraatheid van resultate neem vinnig af, soos die aantal beeldkenmerke en kameraposisies toeneem of soos die ruis in die intree toeneem. Ons is egter oortuig dat verdere ontwikkelinge hierdie stelsel kan verbeter tot so mate dat dit as 'n praktiese en betroubare oplossing vir 'n wye verskeidenheid van werklike datastelle kan dien. 'n Verdere gevolgtrekking is dat hierdie metode groot hulp kan bied aan soortgelyke nie-lineêre probleme, veral dié sonder 'n maklike oplossing.

Acknowledgements

I would like to thank my supervisor Willie Brink for those extra miles he was always willing to walk in providing guidance when things were difficult, as well as my other supervisor Johan Adam du Preez for the insightful learning experience he gave me and his help in finding a direction for this study. Furthermore, I would like to thank the MIH MediaLab for providing funding and an enjoyable environment in which to work, as well as ARMSCOR for providing additional funding. Finally, I would like to express my gratitude to my parents, my siblings and my friends for the positive influence they have on my life.

Contents

1	Introduction	1
1.1	Structure and motion	1
1.2	Probabilistic graphical models	5
1.3	Our approach	5
1.4	Outline of this thesis	6
2	Feature detection and matching	8
2.1	Feature detection with SIFT	8
2.2	Description and matching with SIFT	11
2.3	Other feature detectors and descriptors	11
2.4	Detecting outliers with RANSAC	14
2.5	Extension to multiple images	16
3	Multiple view geometry	18
3.1	Homogeneous coordinates	19
3.2	The pinhole camera model	20
3.3	Camera Calibration	20
3.4	Triangulation	21
3.5	Epipolar geometry	23
3.6	Calculating the essential matrix	25
3.7	Estimating a camera pair	27
3.8	Extension to multiple cameras	28
3.9	Summary	30
4	Basic concepts in probability	32
4.1	Probability distributions	32
4.2	Discrete probability tables	34
4.3	Multivariate Gaussian distributions	39
4.4	Transformations on Gaussian distributions	41
4.5	Sigma point parameterisation	42
5	Probabilistic graphical models	48
5.1	Hamming code example	48
5.2	Bayes networks	49
5.3	Cluster graphs	50
5.4	Message passing	54
5.5	Belief propagation	54

5.6	Loopy belief propagation	56
5.7	Example of solving a Hamming code	59
6	Our probabilistic formulation	61
6.1	The parameters of the system	61
6.2	The dependencies within the system	62
6.3	Nonlinear projective geometry	63
6.4	Linearising projective geometry	64
6.5	Building a cluster graph	68
6.6	Finding the posterior distribution	69
7	Experimental results	72
7.1	Sensitivity to inaccurate priors	73
7.2	Sensitivity to measurement noise	75
7.3	Sensitivity to outliers	77
7.4	Real-world examples	78
8	Conclusions and future work	82
8.1	Future work	83
	List of References	85

Chapter 1

Introduction

The aim of this study is to consider two somewhat separate fields, namely computer vision and probabilistic graphical models, in order to parameterise and solve the well known structure and motion problem probabilistically. In this chapter we outline some of the literature pertinent to this thesis and give an outline and a rationale behind our proposed system.

1.1 Structure and motion

We, as humans, can infer spatial information about our environment and our current position within that environment, from moving through the scene and seeing parts of it. Similarly, in the computer vision literature the so-called structure and motion (or structure from motion) problem asks for simultaneously generating a 3D reconstruction of a scene (the structure) and finding camera positions within the reconstructed scene (the motion) from 2D images. A solution should therefore be a system that takes as input an unordered set of images capturing a scene or object from different poses and generates as output a 3D reconstruction of the scene or object and camera poses related to the images.

The strength of such a system is that a 3D model can be created from data easily or inexpensively obtained. Hobbyists may capture an object using a digital camera or a camera phone, as seen in Figure 1.1, and large image sets of entire cities, as seen in Figure 1.2, can be aggregated from image sharing websites.

These sorts of computer vision capabilities are becoming increasingly popular due to the simplicity and inexpensive nature of obtaining high quality digital images. Structure and motion can aid a variety of applications such as

- 3D graphics for motion pictures, gaming and virtual worlds,
- mapping and archiving geography, cities, or heritage sites,
- CAD based prototyping, and
- manufacturing using 3D printing.



Figure 1.1: An example of reconstructing an object using structure and motion. The result is obtained from an automated process as part of the software package Autodesk 123D Catch [1, 2].

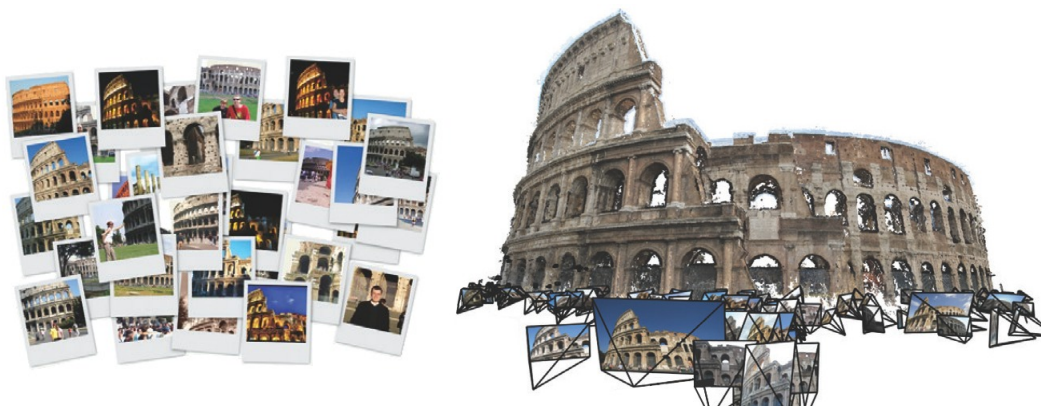
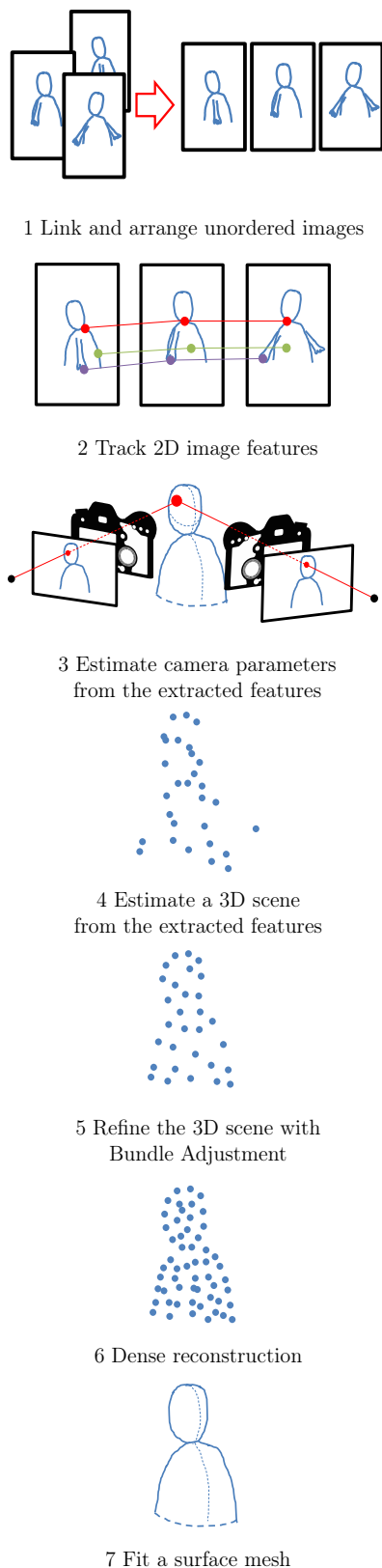


Figure 1.2: An example of a large-scale structure and motion problem. This example features the Colosseum in Rome which was reconstructed from photos aggregated from the image hosting website Flickr [3].

The structure and motion problem has received a lot of attention in recent years with projects such as Microsoft's Photo Tourism [4], Google's Photo Tours [5], and Apple's Flyover 3D Maps [6] as a result. Users are also able to conveniently create models with products such as 123D Catch [1], 3DF ZEPHYR [7] and Neitra 3D Pro [8].

The problem itself is not that straightforward to solve. There is an interdependency between structure and motion (the one cannot be estimated in isolation from the other). The absolute scale of a captured scene is not inherent in images (consider the similarity between photos of a scaled-down car replica captured with a nearby camera and photos of a normal car captured from further away). Image data can be ambiguous and may lead to geometric inconsistencies if points in different images are not matched correctly.

The underlying mathematical theory that is used to model and solve the structure and motion problem is based on multiple view geometry, and is established in the literature by Hartley and Zisserman [9]. To implement a structure and



motion system the following pipeline is typically constructed (Figure 1.3):

1. organise the input images by identifying overlapping images,
2. track the same underlying 3D points, or features, over multiple images,
3. estimate the camera parameters for each image by applying multiple view geometry principles,
4. triangulate the tracked features to estimate a point cloud,
5. use bundle adjustment to refine the 3D scene and camera parameters,
6. use geometric constraints to estimate a more dense point cloud, and
7. fit a surface mesh on the dense point cloud.

This approach is feature based, which means that the data we use to establish the geometry of the system takes the form of matched image features obtained through feature extraction and matching algorithms such as SIFT [10].

Step 1 is only necessary for an unordered image set. The methods used to find overlapping images range from a poorly scaling direct approach where similar features are matched across all images, to a more advanced system that builds a feature tree and querying the tree in order to find similarity information among the images [11]. Additional information, such as geotags for large-scale reconstructions, can also be used to aid this process [12].

In step 2 some feature detection and matching algorithm is used to find matching points between overlapping images, i.e. projections corresponding to the same 3D point. Furthermore, incorrect matches can be removed with algorithms such as RANSAC [13].

Figure 1.3: The structure and motion pipeline

In steps 3–4 multiple view geometry is used to find a set of camera poses by constraining the matching features to be projections of the same 3D point. These 3D points are also found through triangulation. Note that scaling, rotation and all other similarity transformations on the system as a whole do not affect the values of the projections and, therefore, the reconstructed parameters can only be estimated up to a similarity ambiguity.

Bundle adjustment, step 5, refines the structure and motion result by setting up a nonlinear optimisation with parameters as the 3D points and the camera poses and the cost function as the reprojection error. This error is measured as the squared distance between a 2D projection of a 3D point using the camera parameters and the measured 2D location of that feature, summed over all 3D points and camera parameters. The Levenberg-Marquardt algorithm [14] is often used to solve this optimisation problem.

Steps 6–7 are additional enhancements that can be applied if a dense reconstruction is needed for an application such as CAD or 3D printing.

Some landmarks in the structure and motion literature include the Photo Tourism project [15] which presents a system for estimating a scene from large unorganised collections of images and allows interactive browsing of the result. Furthermore, Building Rome in a Day [16] is a system that reconstructs large-scale scenes from unorganised collections of images aggregated from image hosting sites. For instance, the Rome dataset consists of roughly 150,000 images. The key contribution of this work is “a new, parallel distributed matching system that can match massive collections of images very quickly and a new bundle adjust software that can solve extremely large nonlinear least-squares problems.” Microsoft recently developed First-person Hyperlapse Videos, a structure and motion system for first person video in order to smooth out jittery motion by allowing a damped motion to recapture the generated scene [17].

Even though the structure and motion problem is by now well researched and has most of its early obstacles resolved, we find a probabilistic approach to be an important addition to the literature. A probabilistic solution provides additional insight into the system such as the confidence of estimated parameter values. It can be easily expanded to include additional logic such as the penalisation of conflicting parameters. Also, it can be integrated with other probabilistic systems to form very complex systems.

We also focus on a general approach that can capture the variables and inter-relationships of a system with minimum knowledge about the system. Bundle adjustment, on the other hand, is quite complex and tailor-made. It is derived from in-depth knowledge about all projective transformations involved [18]. Furthermore, in order to avoid linearisation problems, it requires initial estimates that are already somewhat accurate [18].

1.2 Probabilistic graphical models

The reason for our focus on a probabilistic approach to the structure and motion problem is twofold. Firstly we want to investigate the extent to which a graphical model approach can solve a general nonlinear problem; and secondly, we want to investigate the possible advantages that a probabilistic approach can add to a structure and motion system. The theory and approaches introduced in this thesis will therefore be kept quite generic in order to remain relevant and easily tailorable for similar problems.

The goal of graphical models is to provide an approach to deal with uncertainty within a large system using probability theory. With this approach decisions are made by calculating the probability of a particular outcome of a system based on incomplete prior knowledge about the parameters of the system. Typical applications for graphical models include medical diagnosis and troubleshooting, where the observation of symptoms can lead to knowledge about the cause [19]; image denoising [20]; image segmentation and classification [21]; and traffic analysis [22].

A graphical model is constructed from probabilistic information about the variables of a system. This includes the dependencies between variables and the estimated probability distributions over some of the variables. After the available information is integrated in a graph structure, such as a cluster graph or factor graph, belief propagation algorithms can be applied on the graph in order to find the posterior belief of the system, i.e. the probability distribution over all the variables after taking all information into account. Finally, if needed, the most likely state for every variable can be extracted from this distribution.

1.3 Our approach

We want to describe the dependencies between the structure and motion parameters probabilistically and provide information about the possible values of the variables. The system consists of spatial parameters (3D point coordinates and 6DoF camera poses) which are continuous. This leads us to the choice of parameterising the system with Gaussian variables. The reason is that Gaussians are mathematically convenient and well defined within the probabilistic graphical model literature. Also, it would be very difficult to implement the exact types of distributions resulting from the nonlinearities of the system.

In light of the nonlinear relationships between the parameters, and our focus on a general solution that can be tailored for a wide range of applications, we decide to use sigma point formulations for linearisation [23]. Sigma point parameterisation can linearise relationships between variables without in-depth knowledge about those relationships, such as the associated Jacobian matrix.

Through consideration of both the computer vision and the graphical models literature, we have created the following pipeline for our probabilistic approach:

1. we find initial estimates for the parameters of the structure and motion problem by applying steps 1–5 from Figure 1.3,
2. these estimated parameters are then used as prior knowledge for the probabilistic system by fitting wide Gaussian distributions over them,
3. all dependencies within the system are then captured and linearised using sigma points,
4. a graphical model, specifically a cluster graph, is then built from these dependencies,
5. the projected points (the matched features) are then “observed” by assigning a high certainty to their values,
6. belief propagation is then run on the system to find the posterior beliefs of the parameters involved.

The result obtained from this pipeline is therefore not a strict solution to the problem, but rather a random distribution representing the system. Probabilistic graphical models are readily extended to allow for additional parameters or to be merged with other systems. This approach would therefore allow structure and motion to be an addition to other systems. Such applications might include object segmentation or collision detection for autonomous vehicles.

1.4 Outline of this thesis

The next two chapters of this thesis are dedicated to computer vision and set out to explain the structure and motion problem. After this the focus switches to probability theory and probabilistic graphical models. The remaining part of the thesis is then dedicated to merging the two parts in order to create a probabilistic structure and motion system.

In order to extract the multiview geometry between images and estimate 3D structure and motion, we need to find similarities between images. Therefore our discussion in Chapter 2 (*Feature detection and matching*) starts with feature detection and matching algorithms, outlier removal on these feature matches and ends with a discussion on finding neighbouring images from an unordered image set.

In Chapter 3 (*Multiple view geometry*) we discuss the pinhole camera models and the projection of 3D space to 2D space. The discussion then progresses to triangulation, epipolar geometry and the 8-point algorithm used to calculate initial estimates of camera poses and a 3D model (consisting of 3D points). The chapter ends with a working structure and motion pipeline.

For the next two chapters our focus shifts towards probability theory and graphical models. Chapter 4 (*Basic concepts in probability*) provides tools

necessary to implement a probabilistic graphical model. This includes mathematical operations for discrete and Gaussian distributions and linearising non-linear transformations using sigma points.

In Chapter 5 (*Probabilistic graphical models*) we implement a working probabilistic graphical model design. We discuss how a system of random variables is broken up into parts (clusters), how these clusters are connected into a graph structure according to dependencies between variables, and how belief propagation can be applied to find a posterior distribution for the system.

The theory discussed is then combined in Chapter 6 (*Our probabilistic formulation*) in order to build a cluster graph for the structure and motion problem and run belief propagation on it. An example of a working 2D structure and motion example is provided as an accessible illustration of the complete system.

Finally, in Chapter 7 (*Experimental results*) we first test and quantitatively evaluate the system using synthetic data, and then run the system using two real-world datasets. We draw final conclusions and discuss possibilities for future study in Chapter 8 (*Conclusions and future work*).

Chapter 2

Feature detection and matching

Before we can use multiview geometry for the purpose of finding an initial structure and camera poses we need to relate the images of a given set. A key in finding this relationship is the ability to find 2D points in different images which correspond to the same point in 3D space.

One approach towards finding such correspondences is feature detection and matching, of which the scale invariant feature transform (SIFT) [10] has become a gold standard. The idea is to find prominent points in all images and then use descriptor vectors to match up these points between different images.

In this chapter we discuss feature detection and matching, specifically with regards to SIFT. We further discuss how to remove incorrect matches through random sample consensus (RANSAC). In Chapter 3 the discussion progresses towards multiview geometry and methods for finding an initial model for the 3D structure and camera poses.

2.1 Feature detection with SIFT

The scale invariant feature transform (SIFT) published by Lowe in 1999 [10] can be used to find interest points in images. The goal is to find points in an image that are likely to be redetected on affine transformations and illumination variations of the same image. Since moderate projective transformation may be approximated by affine transformations, such as in the case where an object is captured from a slightly different camera position, SIFT can be effective on images depicting some physical scene from various viewpoints.

The SIFT algorithm detects features on a given image as follows. We first apply K difference-of-Gaussian (DoG) filters, which act as band pass filters, on a grey scale version of the image to obtain a list of DoG images. Each of these DoG images represents a certain frequency range or image sharpness. Figure 2.1 provides an example.

To then find feature coordinates, we stack the DoG images to form a 3D DoG scale-space and identify the local minima and maxima. If we have a volume of dimension $I \times J \times K$ (where I is the number of DoG filters and $J \times K$ is

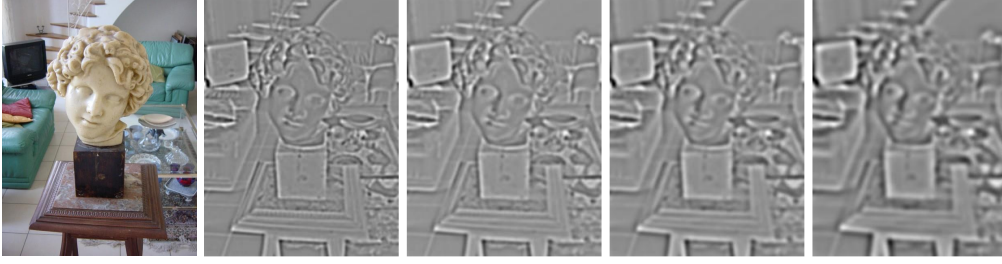


Figure 2.1: The effects of a Difference-of-Gaussian filter. On the left we have the original image and the subsequent images are filtered with a DoG filter with increasing intensity. We investigate local minima and maxima in these DoG filtered images as part of the SIFT feature detection routine.

the image dimensions) we are looking for each occurrence where an element in the middle of a $3 \times 3 \times 3$ slice is the minimum or maximum element within that slice. To simulate zooming and to make SIFT more scale invariant, the feature detection process is repeated on scaled-down versions of the original image.

Also, if sub-pixel accuracy is needed, the features locations can be refined through sub-sampling the DoG scale-space. One method is to use the quadratic Taylor expansion [24]. If \mathbf{x}_0 is an initial feature location within the DoG scale-space and

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0 + \mathbf{h}_0 \quad (2.1)$$

is a sub-pixel refinement of this feature location, we use the second-order Taylor expansion to find a continuous function $D(\mathbf{x})$ to represent the space around \mathbf{x}_0 as

$$D(\mathbf{x}_0 + \mathbf{h}) \approx D(\mathbf{x}_0) + \left(\frac{\partial D(\mathbf{x})}{\partial \mathbf{x}} \right)^T \bigg|_{\mathbf{x}=\mathbf{x}_0} \mathbf{h} + \frac{1}{2} \mathbf{h}^T \left(\frac{\partial^2 D(\mathbf{x})}{\partial \mathbf{x}^2} \right)^T \bigg|_{\mathbf{x}=\mathbf{x}_0} \mathbf{h}. \quad (2.2)$$

The derivative with respect to \mathbf{h} is found by means of finite differences within the discrete DoG scale-space, and we find \mathbf{h}_0 by setting that derivative to zero.

We remove unreliable features where the image gradient is not steep enough or only steep in a single direction, i.e. where a feature lies in a smooth region or where a feature lies on an edge (as opposed to a corner) in the image.

We associate each feature with a dominant angle, to allow the description of features to be independent of image rotation. We determine this angle by inspecting the image gradient around each feature. We find a 16×16 image gradient patch (consisting of gradient magnitude and orientation) around a feature and weigh this patch with a centred Gaussian to reduce the influence of distant elements. Next we build a histogram from this patch where gradient magnitude contributes to orientation bins of equal sizes. Finally, the dominant angle is taken as the peak of the histogram. To improve accuracy, the



Figure 2.2: Interest points obtained through the DoG images: the (x, y) coordinates are extracted by taking the 3D extrema of the stacked DoG filtered images.

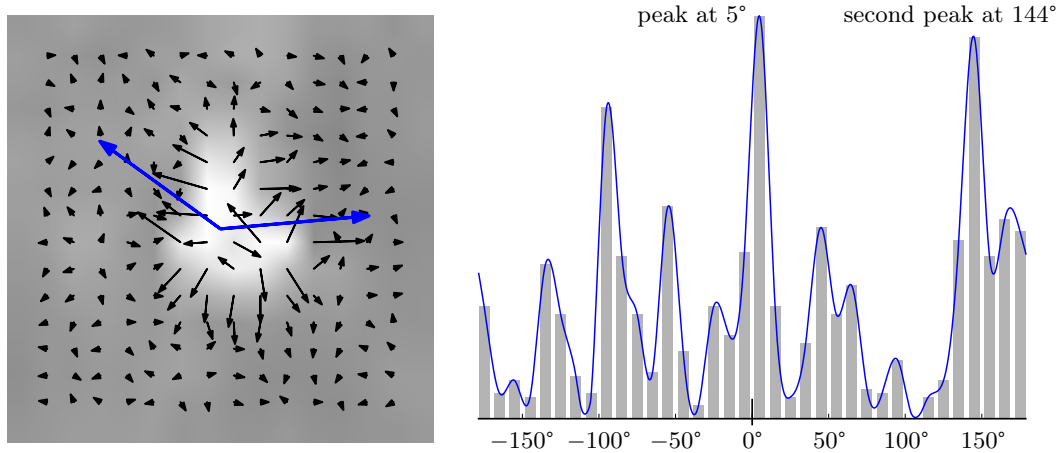


Figure 2.3: Finding the dominant angle of a feature point: on the left is a 16×16 image gradient window around the feature where the blue arrows are the dominant angles of this feature; on the right is a histogram of the gradient (in grey) and an interpolation thereof (in blue).

histogram may be interpolated. As a rule, in the case where a second peak is found at more than 80% of the maximum, we include this feature as an additional feature with the second peak as the dominant angle.

The above process is repeated on different scales of the input image. This allows for features more prominent on a smaller scale (where neighbouring influences are taken from further away) to surface.

2.2 Description and matching with SIFT

A SIFT descriptor is a 128-dimensional vector used to describe a feature. To measure how closely two features match, we compute the Euclidean distance between their descriptors.

SIFT descriptors are calculated as follows. First we surround each feature point with a rectangular patch of size and rotation determined by the feature's scale and dominant angle. This patch is divided into 16×16 pixels for which we obtain values by sampling the corresponding points in the underlying image.

A descriptor is then calculated from the histogram response of the image gradient of the 16×16 patch, weighted by a centred Gaussian. The gradient is divided into 4×4 blocks and for each of these blocks we calculate a histogram of 45° bins (8 bins in total) with weights determined by the gradient magnitude. Finally the responses from all the histograms are concatenated and then normalised.

A descriptor is therefore a comparable summary of the spatial information around a feature point. We can now use the descriptors to match features between two images, as shown in Figure 2.5. The standard approach is to apply a nearest neighbour search. Heuristics such as Fast Approximate Nearest Neighbours (FLANN) are available to reduce search complexity [25].

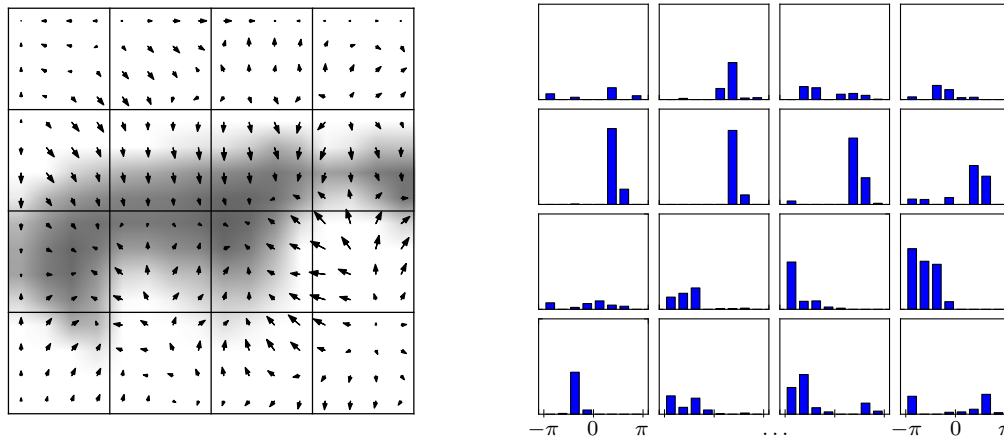


Figure 2.4: We build a SIFT descriptor by taking histogram responses from 4×4 sampled gradient points around a feature. The descriptor is the resulting histograms concatenated into a single 128-dimensional vector.

2.3 Other feature detectors and descriptors

As is evident from our explanation of SIFT, feature detection and feature description are two distinct phases that need to be applied before feature



Figure 2.5: SIFT feature matches between an image pair. Only 20% of the matches, randomly chosen, are displayed in order to reduce clutter. Some of the matches are clearly not true correspondences.

matching can take place. We now discuss and compare some of the alternative detecting and descriptor methods. A summary of popular feature detectors and descriptors can be seen in Table 2.1.

We categorise feature detection into two approaches: blob detection and corner detection. Blob detectors detect local extrema of the response of scale-space type filters, such as the DoG filter used by SIFT. The main aim of SURF is to offer a computationally efficient alternative to SIFT detection. As a result the DoG filter is estimated by applying box filters with an efficient implementation using integral images, as a trade-off in accuracy. Similarly, the CenSurE feature detector uses this speeded up filtering process, but without using octaves of different sizes, i.e. the scale-space used when filtering is always the full image size. This is an attempt to find more stable features at the higher levels of the scale-space pyramid.

There is no clear definition for a corner in the computer vision literature, but some define it as the point where two nonparallel edges meet, or ought to meet. Corner detection usually outperforms blob detection in terms of efficiency and is mostly used in systems where speed is important. Corner detection algorithms have been around as early as the 1980s with the Moravec corner detector [34] and an improvement by Harris and Stephens [35] which both use the image gradient in two directions to determine a corner. More modern equivalents such as FAST and AGAST use the following test to find a corner: if a pixel is either dimmer or brighter within a certain threshold than n continuous pixels on a circle of radius 3 pixels (estimated using the Bresenham's circle algorithm), that pixel is classified as a corner. FAST uses a machine learning approach to classify a corner by testing only a few pixels, and AGAST offers a speedup for the same concept. BRISK and ORB on the other hand add scale-space analysis to this approach, mimicking SIFT's approach to achieve scale invariance.

	Detector	Descriptor
SIFT, 1999 [10] Scale-invariant feature transform	blob-like	nonbinary
SURF, 2006 [26] Speeded up robust features	blob-like	nonbinary
CenSurE, 2008 [27] Center surround extremas for realtime feature detection and matching	blob-like	-
FAST, 2010 [28] Features from accelerated segment test	corner	-
AGAST, 2010 [29] Adaptive and generic corner detection based on the accelerated segment test	corner	-
BRIEF, 2010 [30] Binary robust independent elementary features	-	binary
BRISK, 2011 [31] Binary robust invariant scalable keypoints	corner	binary
ORB, 2011 [32] Oriented FAST and rotated BRIEF	corner	binary
FREAK, 2012 [33] Fast retina keypoint	-	binary

Table 2.1: A list of popular feature detectors and descriptors. The detection and description methods are explained in more detail in the rest of Section 2.3.

SIFT and SURF have nonbinary descriptors in the form of arrays of floating point numbers. With SURF a circular region around the feature point is chosen (similar to SIFT). This region is then divided into 4×4 subregions of which each is convolved with two orthogonal Haar wavelets. The descriptor is built up from information such as the summation and absolute summation of filter responses on each of the subregions. BRISK, ORB, BRIEF and FREAK on the other hand have binary descriptors. The main idea is to use a sample pattern to choose pixel pairs around the feature point and compare the intensities within each pair to form a binary string. A pixel pair approach is also used to find the orientation of the feature. These binary strings can be compared by using the Hamming distance, which can be executed very fast using the XOR operator.

A comparative study by Canclini [36] set out to “provide an up-to-date detailed, clear, and complete evaluation of local feature detector and descriptors, focusing on the methods that were designed with complexity constraints, pro-

viding a much needed reference for researchers in this field.” The study found the following. In terms of speed, corner detectors have a vast advantage over blob detectors, of which SIFT is considerably slower than SURF. With regards to invariance to transformations, the results varied according to the type of transformation. Although SIFT is not the leading detector in any of the transformation tests, its performance was fairly constant where the other detectors varied widely according to particular transformations. With regards to descriptor speed, the binary descriptors outperformed the nonbinary descriptors considerably. Matching accuracy was measured by testing the validity of each match with an underlying ground truth 3D structure. The nonbinary SIFT matches proved to be the most accurate. However, the other nonbinary descriptor SURF underperformed in comparison with ORB and BRISK.

Since the requirements for features used in this study lean more towards accuracy than speed, we opt to make use of SIFT. It should be stressed, however, that the rest of the discussion in this thesis is essentially independent of the choice of feature detector/descriptor.

2.4 Detecting outliers with RANSAC

Because of the ambiguous nature of image data, it is typical for any feature detector and matcher to return a number of incorrect matches between two images. We will now discuss Random Sample Consensus (RANSAC) as a technique to classify inliers (datapoints that are seemingly correct) and outliers (datapoints that are seemingly incorrect) from a given set. It can be used to find and discard erroneous matches from our set of SIFT matches.

RANSAC [13] is a method to obtain a model from data which contains erratic noise. Its goal can best be explained with the following simple example. Suppose we try to fit a line through a set of 2D points containing a large number of outliers, as presented in Figure 2.6. Naive methods for estimating a line through this data, such as finding a least-squares solution, would yield poor results since the outliers will have a drastic influence on the estimated model. With RANSAC, we simultaneously classify inliers and fit a model with those inliers. Note that RANSAC is most effective for data with minimal noise in the inliers and erratic outlier behaviour (so that the structure of the outliers cannot be explained by a single model).

Suppose we are given a set of observations that fit a model well (inliers), and a set of erratic data (outliers). Suppose further that we have a means of fitting a model to a minimal subset of the data, as well as an error metric that can be used to split the data into inliers (close to the fitted model) and outliers (far from the fitted model). The RANSAC philosophy is to repeatedly sample and fit a model from a mix dataset, and take the model that yields the largest set of inliers.

If k is the minimum number of datapoints needed to fit the model in question, then there are $\frac{n!}{k!(n-k)!}$ models that can be fitted. It is too expensive to search

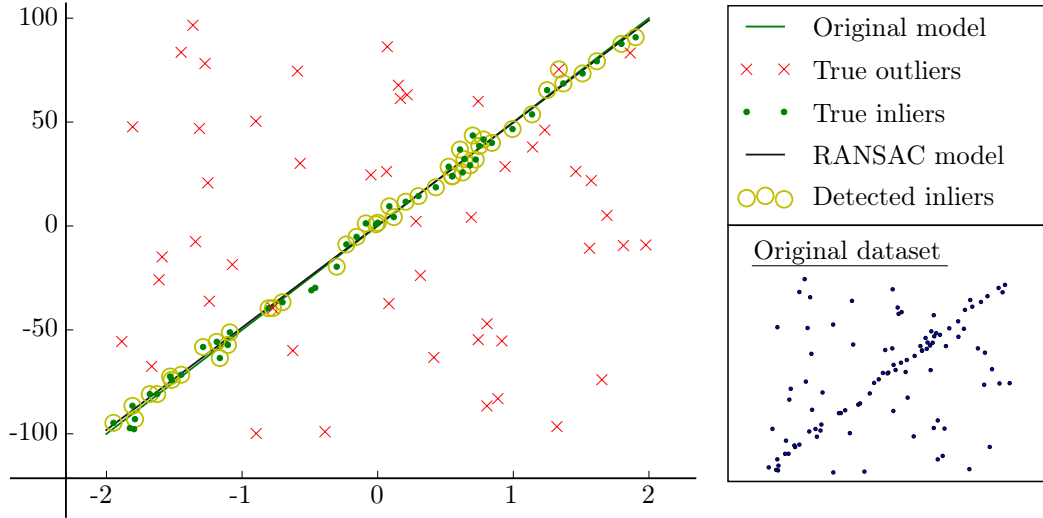


Figure 2.6: An example of RANSAC classifying inliers and outliers in a noisy dataset. Our original model is the line $y = 50x$. The dataset contains 100 points of which 50 are randomly distributed on the original model line, with added Gaussian noise, and the other 50 are randomly distributed over a region in \mathbb{R}^2 . The model obtained by RANSAC is a good approximation of the original model. Four inliers were incorrectly classified as outliers, and two outliers were incorrectly classified as inliers. Further investigation revealed that the placement of these misclassified outliers makes them indistinguishable from the true inliers.

through all possible combinations to find the best model, so the RANSAC heuristic rather fits only N models with k randomly chosen datapoints and chooses the best model out of that pool. This is based on the assumption that a set of inliers will be found within those N iterations and that those inliers will produce an accurate model which fits the other inliers. The RANSAC algorithm is laid out in more detail in Algorithm 1.

In the case where the algorithm's input \mathcal{S} is a dataset of feature matches from an image pair we can refer to the theory on multiview geometry (discussed in Chapter 3) for the following choices regarding the other input parameters: set the function $f(\mathcal{X})$ as the 8-point algorithm (with a fundamental matrix F as output), set the error metric $e(\mathbf{x}, F)$ as the Sampson distance, set k as 8, and set the error threshold t to a value of around 5 (shown in experiments to yield reasonable results). Figure 2.7 shows an example of using RANSAC on feature matches.

Hartley and Zisserman [9] suggest the following choice for the number of iterations N :

$$N = \frac{\log(1 - p)}{\log(1 - w^k)}, \quad (2.3)$$

where p is the chosen probability that the algorithm will result in a good solution, w is an estimated probability of choosing an inlier if a point is drawn

at random and k is the minimum number of points needed to fit the chosen model.

Although RANSAC is widely implemented as described, some improvements have been suggested:

- a “bail-out” test to ensure that the main loop in line 4 can be prematurely terminated if a set of inlier matches has likely been found [37];
- PROSAC [38] as an adaptation where the sample set in line 5 is not drawn according to a uniform random distribution, but according to a distribution biased towards good matching scores;
- SCRAMSAC [39] as an adaptation where each matching feature pair is tested to have neighbouring features that are consistent in both images.

Algorithm 1: RANSAC

Input: dataset \mathcal{S} ; function $f(\mathcal{X})$ to fit a model; function $e(\mathbf{x}, M)$ to determine the distance (or error) between a datapoint and a model; minimum number k of points needed to fit the model; error threshold t ; number N of iterations

Output: inlier set \mathcal{I} ; fitted model M

```

1:  $M := \text{null}$ 
2:  $\mathcal{I} := \{\}$ 
3:  $e := \infty$ 
4: repeat  $N$  times
5:    $\mathcal{X}' := \text{random } k \text{ points from } \mathcal{S}$ 
6:    $M' := f(\mathcal{X}')$ 
7:    $e' := \text{the sum of } e(\mathbf{x}, M') \text{ over all } \mathbf{x} \in \mathcal{S}$ 
8:    $\mathcal{I}' := \text{the set of all datapoints } \mathbf{x} \text{ from } \mathcal{S} \text{ where } e(\mathbf{x}, M') < t$ 
9:   if  $|\mathcal{I}'| < |\mathcal{I}|$  or  $(|\mathcal{I}'| = |\mathcal{I}| \text{ and } e' < e)$  then
10:     $M := M'$ 
11:     $\mathcal{I} := \mathcal{I}'$ 
12:     $e := e'$ 
13:   end if
14: end repeat

```

2.5 Extension to multiple images

Given an order by which multiple images connect to one another, such as a connectivity graph or a list of sequential images, we can find feature matches across all of those images using the following logic. Consider a sequence of



Figure 2.7: Outlier removal with RANSAC on SIFT feature matches between an image pair. Only 20% of the matches, randomly chosen, are displayed in order to reduce clutter.

images $\{I_1, \dots, I_n\}$ and corresponding feature sets $\{\mathcal{F}_1, \dots, \mathcal{F}_n\}$, with a feature taken from the i^{th} feature set denoted as $\mathbf{f}_i^a \in \mathcal{F}_i$. If we find pairwise matches between each neighbouring images (e.g. $\{\mathcal{I}_1, \mathcal{I}_2\}$, $\{\mathcal{I}_2, \mathcal{I}_3\}$, \dots $\{\mathcal{I}_{n-1}, \mathcal{I}_n\}$), then if \mathbf{f}_i^a is matched with \mathbf{f}_j^b in one pair, and \mathbf{f}_j^b is match with \mathbf{f}_k^c in another pair etc., we take these features as representing the same underlying point. We thus obtain feature correspondences over multiple images.

If we follow this approach for an image set with loopy connectivity there is a possibility that a feature within one image might not match up to itself when completing a loop back to that particular image. This is due to the fact that we cannot be assured that a feature match is correct. We can either break such a feature matching chain at some chosen point or discard all the matches in the chain.

If the image set is unordered, it might be necessary to determine the connectivity between images. A naive approach would be to compare all the feature sets \mathcal{F}_i with one another and construct a connectivity graph or a maximum spanning tree. This approach scales quadratically with the number of images and is therefore not feasible for large image sets. Another, more scalable, approach is through the use of a vocabulary tree [11], a type of lookup system which can be used to find similar images from a set of training data.

Chapter 3

Multiple view geometry

In this chapter we describe the geometry behind the formation of digital images, and show how to obtain an initial 3D scene and camera poses from feature correspondences. In the chapters that follow we will discuss methods for refining such an initial estimate using probabilistic graphical models.

The theory in this chapter will provide the necessary tools to model camera poses and a 3D scene. This includes

- projective geometry along with the homogeneous representation of points and lines,
- the pinhole camera model which models the projection of 3D space to image coordinates, and
- the triangulation of a 3D point from multiple camera poses and matching image points.

Furthermore we provide the theory and tools to estimate structure and motion from images. We explain

- epipolar geometry (the geometry behind stereo images) along with the essential matrix as a parameterisation thereof,
- how to obtain the epipolar geometry for an image pair using feature matches,
- how to obtain camera poses from epipolar geometry, and
- finally how to extend this stereo approach to multiple images.

We end this chapter with a structure and motion pipeline derived from the various tools, along with an example.

3.1 Homogeneous coordinates

We explain the use of homogeneous coordinates as an introduction to projective geometry. Homogeneous coordinates represent points and lines with benefits over Euclidean coordinates of having common operations described by simple vector and matrix operations. Homogeneous vectors are denoted with a tilde such as $\tilde{\mathbf{x}}$.

Points and lines in \mathbb{R}^2 space can be transformed to homogeneous coordinates in the \mathbb{P}^2 projective space as follows:

$$\text{point } \mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^T \text{ to } \tilde{\mathbf{x}} = \begin{bmatrix} x & y & 1 \end{bmatrix}^T, \quad (3.1)$$

$$\text{line } ax + by + c = 0 \text{ to } \tilde{\mathbf{l}} = \begin{bmatrix} a & b & c \end{bmatrix}^T. \quad (3.2)$$

Similarly a point in \mathbb{R}^3 transforms to an element of \mathbb{P}^3 as

$$\text{point } \mathbf{X} = \begin{bmatrix} x & y & z \end{bmatrix}^T \text{ to } \tilde{\mathbf{X}} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T. \quad (3.3)$$

Note that for any homogeneous vector, a scalar multiplication thereof still describes the same point. Also, homogeneous coordinates can describe points and lines at infinity (such as point $\tilde{\mathbf{x}} = [x \ y \ 0]^T$ and line $\tilde{\mathbf{l}} = [0 \ 0 \ c]$), which cannot be expressed by Euclidean coordinates.

For points and lines in their homogeneous forms, the following hold:

- two homogeneous points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ represent the same point if

$$\tilde{\mathbf{x}} \times \tilde{\mathbf{x}}' = \mathbf{0}, \quad (3.4)$$

- and similarly two homogeneous lines $\tilde{\mathbf{l}}$ and $\tilde{\mathbf{l}}'$ are equivalent if

$$\tilde{\mathbf{l}} \times \tilde{\mathbf{l}}' = \mathbf{0}, \quad (3.5)$$

- a point $\tilde{\mathbf{x}}$ lies on the line $\tilde{\mathbf{l}}$ if and only if

$$\tilde{\mathbf{x}}^T \tilde{\mathbf{l}} = 0, \quad (3.6)$$

- the intersection of two lines $\tilde{\mathbf{l}}$ and $\tilde{\mathbf{l}}'$ is the point

$$\tilde{\mathbf{x}} = \tilde{\mathbf{l}} \times \tilde{\mathbf{l}}', \text{ and} \quad (3.7)$$

- and the line joining two points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ is

$$\tilde{\mathbf{l}} = \tilde{\mathbf{x}} \times \tilde{\mathbf{x}}'. \quad (3.8)$$

Furthermore, we can apply a homography that includes rotation, translation, scaling, skewing, mirroring or indeed any projective transformation to a point \mathbf{x} with simple matrix multiplication. A homography H can be applied to a point \mathbf{x} as

$$\tilde{\mathbf{x}}' = H\tilde{\mathbf{x}}, \quad (3.9)$$

where $\tilde{\mathbf{x}}'$ is the transformed $\tilde{\mathbf{x}}$, $\tilde{\mathbf{x}}$ is a point in \mathbb{P}^{n-1} and H is an $n \times n$ nonsingular matrix.

3.2 The pinhole camera model

The pinhole camera model provides a projection from 3D space onto a 2D image plane and approximates the action of a real camera. A diagrammatic representation can be seen in Figure 3.1. The projection of a 3D point \mathbf{X} to an image point \mathbf{x} can be performed by a camera matrix P as follows [9]:

$$\tilde{\mathbf{x}} = P\tilde{\mathbf{X}}, \quad (3.10)$$

with P a 3×4 camera projection matrix which can further be expanded as

$$P = KR \begin{bmatrix} I & | & -\mathbf{C} \end{bmatrix}. \quad (3.11)$$

The camera centre \mathbf{C} and the 3×3 rotation matrix R govern the pose of the camera and can be classified as the extrinsic camera parameters. The calibration matrix K , containing the intrinsic camera parameters, on the other hand is unchanged by the pose of a camera, but may vary between different cameras.

The goal of the calibration matrix is to provide a mapping for a point \mathbf{x} on the projection plane to a point \mathbf{X} on the image plane:

$$\tilde{\mathbf{x}} = K\tilde{\mathbf{X}}. \quad (3.12)$$

It has the form

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (3.13)$$

with f_x and f_y the pixel focal lengths (in the x - and y -directions respectively) and (c_x, c_y) the image centre.

We can obtain the intrinsic parameters of a camera through the camera calibration procedure discussed in the next section. For most of the discussion further on we work with camera matrices stripped of K , as to focus on obtaining the extrinsic camera parameters (i.e. the poses of the cameras). We use the normalised camera matrix

$$\bar{P} = R \begin{bmatrix} I & | & -\mathbf{C} \end{bmatrix}, \quad (3.14)$$

with normalised coordinates (i.e. points on the projection plane)

$$\tilde{\mathbf{x}} = \bar{P}\tilde{\mathbf{X}}. \quad (3.15)$$

3.3 Camera Calibration

We can find the intrinsic camera parameters in K through the camera calibration procedure proposed by Zhang [40]. Implementations of this procedure are

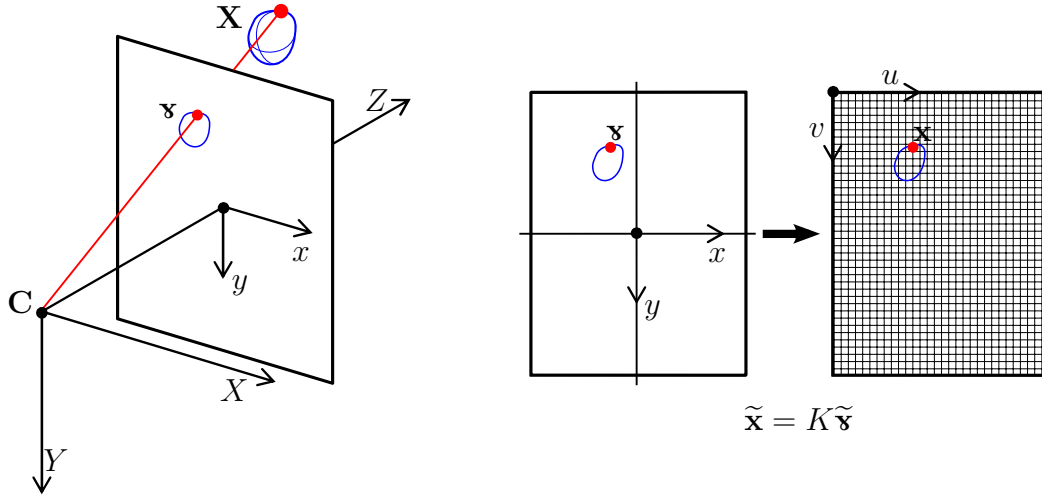


Figure 3.1: The pinhole camera model $\tilde{\mathbf{x}} = P\tilde{\mathbf{X}}$. On the left we have the coordinate system of the pinhole camera model and the projection from 3D space to 2D space as $\tilde{\mathbf{x}} = P\tilde{\mathbf{X}}$. On the right we transform the projection plane to the image plane (measured in pixels) with the calibration matrix K .

freely available through various libraries such as OpenCV [41] and the Camera Calibration Toolbox for Matlab [42].

The idea behind this procedure is to take pictures of a known chequerboard from different poses, such as in Figure 3.2. All corner points on the chequerboard are then detected and tracked over all the images. The intrinsic parameters of the camera are obtained by optimising according to the following constraints. The detected points are projections from 3D points on a flat plane with known distances between them and the intrinsic parameters remain constant regardless of camera or chequerboard movement. A complete description of this procedure can be found in the paper by Zhang [40].

3.4 Triangulation

In the case where we have multiple cameras with projections of an unknown 3D point, as well as full knowledge of all the camera matrices, we can triangulate these projections to find the coordinates of the 3D point. We now investigate the constraints imposed by the projection of a single point regarding a single camera, and from there we determine what is needed to reverse this projection.

With a known projection $\mathbf{x} = [x \ y \ 1]^T$, a known camera pose P and an unknown point \mathbf{X} , we have the homogeneous equality

$$\tilde{\mathbf{x}} = P\tilde{\mathbf{X}}. \quad (3.16)$$

We can reorganise these parameters in a list of linear equations and solve for $\tilde{\mathbf{X}}$ as follows. Equation 3.16 represents a homogeneous equivalence, and we

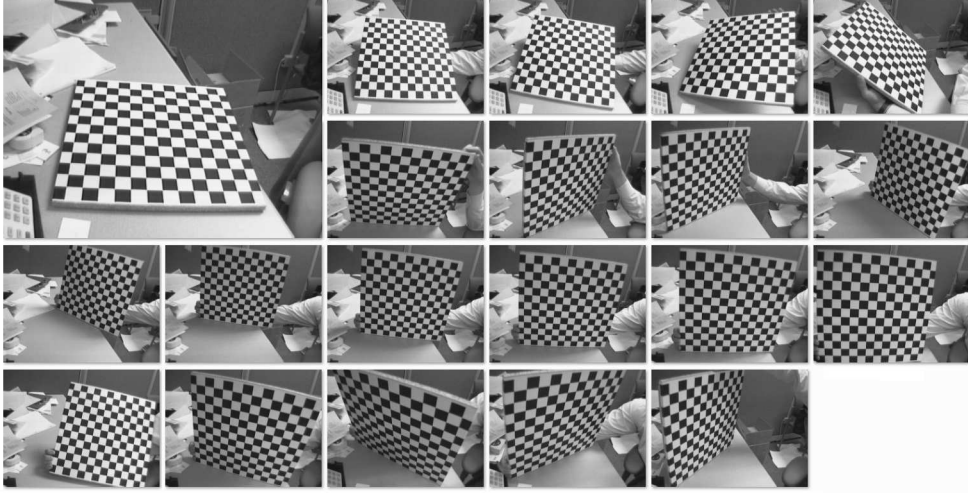


Figure 3.2: A calibration example provided by the Camera Calibration Toolbox for Matlab [42]. As input we have a set of 20 images of a chequerboard with $30\text{mm} \times 30\text{mm}$ squares. The procedure calculates the intrinsic camera parameters and with that information the chequerboard poses can be calculated. Images from [42].

use Equation 3.4 to express the strict equality

$$\tilde{\mathbf{x}} \times \tilde{p}\tilde{\mathbf{X}} = \mathbf{0}. \quad (3.17)$$

A set of linear equations can now be derived as

$$\begin{aligned} y\mathbf{p}_3^T\tilde{\mathbf{X}} - \mathbf{p}_2^T\tilde{\mathbf{X}} &= 0, \\ \mathbf{p}_1^T\tilde{\mathbf{X}} - x\mathbf{p}_3^T\tilde{\mathbf{X}} &= 0, \\ x\mathbf{p}_2^T\tilde{\mathbf{X}} - y\mathbf{p}_1^T\tilde{\mathbf{X}} &= 0, \end{aligned} \quad (3.18)$$

where \mathbf{p}_n^T is the n^{th} row of \tilde{p} .

Note that there is redundancy in this system (any one equation is a linear combination of the other two) and there are not enough restrictions to solve for $\tilde{\mathbf{X}}$ uniquely. We therefore need to extend the system by considering a minimum of two different camera projections of \mathbf{X} .

By extending Equation 3.18 with a set of camera poses $\{\tilde{p}, \tilde{p}', \dots, \tilde{p}^{(n)}\}$, along with $\{\tilde{\mathbf{x}}, \tilde{\mathbf{x}}', \dots, \tilde{\mathbf{x}}^{(n)}\}$ as the respective projections of the same unknown 3D point \mathbf{X} , we have

$$\begin{aligned} \tilde{\mathbf{x}} \times \tilde{p}\tilde{\mathbf{X}} &= \mathbf{0}, \\ \tilde{\mathbf{x}}' \times \tilde{p}'\tilde{\mathbf{X}} &= \mathbf{0}, \\ &\vdots \\ \tilde{\mathbf{x}}^{(n)} \times \tilde{p}^{(n)}\tilde{\mathbf{X}} &= \mathbf{0}, \end{aligned} \quad (3.19)$$

which leads to the system

$$\left\{ \begin{array}{l} y\mathbf{p}_3^T\tilde{\mathbf{X}} - \mathbf{p}_2^T\tilde{\mathbf{X}} = 0, \\ \mathbf{p}_1^T\tilde{\mathbf{X}} - x\mathbf{p}_3^T\tilde{\mathbf{X}} = 0, \\ y'\mathbf{p}_3'^T\tilde{\mathbf{X}} - \mathbf{p}_2'^T\tilde{\mathbf{X}} = 0, \\ \mathbf{p}_1'^T\tilde{\mathbf{X}} - x'\mathbf{p}_3'^T\tilde{\mathbf{X}} = 0, \\ \vdots \\ y^{(n)}\mathbf{p}_3^{(n)T}\tilde{\mathbf{X}} - \mathbf{p}_2^{(n)T}\tilde{\mathbf{X}} = 0, \\ \mathbf{p}_1^{(n)T}\tilde{\mathbf{X}} - x^{(n)}\mathbf{p}_3^{(n)T}\tilde{\mathbf{X}} = 0, \end{array} \right\} \Rightarrow \left[\begin{array}{c} y\mathbf{p}_3^T - \mathbf{p}_2^T \\ \mathbf{p}_1^T - x\mathbf{p}_3^T \\ y'\mathbf{p}_3'^T - \mathbf{p}_2'^T \\ \mathbf{p}_1'^T - x'\mathbf{p}_3'^T \\ \vdots \\ y^{(n)}\mathbf{p}_3^{(n)T} - \mathbf{p}_2^{(n)T} \\ \mathbf{p}_1^{(n)T} - x^{(n)}\mathbf{p}_3^{(n)T} \end{array} \right] \tilde{\mathbf{X}} = A\tilde{\mathbf{X}} = \mathbf{0}. \quad (3.20)$$

Note that in the case of exact measurements, A will have a rank of 3 and $\text{null}(A)$, the null space of A , will be a 1D vector space. This is not a problem since $\tilde{\mathbf{X}}$ is in homogeneous coordinates and therefore also occupies a 1D vector space. We can thus find a suitable $\tilde{\mathbf{X}}$ as a basis for $\text{null}(A)$.

In the case where we have noisy parameters A will have full rank and no basis for $\text{null}(A)$ will exist. We may then estimate $\text{null}(A)$ as a least-squares solution using the singular value decomposition (SVD) [9].

In summary, to obtain a 3D point \mathbf{X} from camera poses and associated projections, we first populate A with the parameters of at least two camera poses and their associated projections, then take $\tilde{\mathbf{X}}$ as a basis for the null space of A , and finally divide the vector by its fourth element to get the 3D coordinates \mathbf{X} (refer to Equation 3.3).

It should be stressed that this technique works only if all the camera poses are known. In the next section we consider the geometry of a two-camera system, which will allow us to extract camera matrices from feature correspondences.

3.5 Epipolar geometry

We want to be able to estimate camera matrices from feature correspondences and then triangulate those correspondences to find a 3D model of the captured scene. We now discuss epipolar geometry, the geometry of stereo vision, along with the essential matrix as a parameterisation thereof. This will be the key to finding camera matrices.

Given an image pair $\{I, I'\}$ and associated camera poses $\{P, P'\}$, we define the epipoles \mathbf{e} and \mathbf{e}' as follows: epipole \mathbf{e} is the projection of the camera centre \mathbf{C}' onto the projection plane of P , that is

$$\tilde{\mathbf{e}} = P\tilde{\mathbf{C}}', \quad (3.21)$$

and similarly epipole \mathbf{e}' is given as

$$\tilde{\mathbf{e}}' = P'\tilde{\mathbf{C}}. \quad (3.22)$$

A visual explanation is given in Figure 3.3.

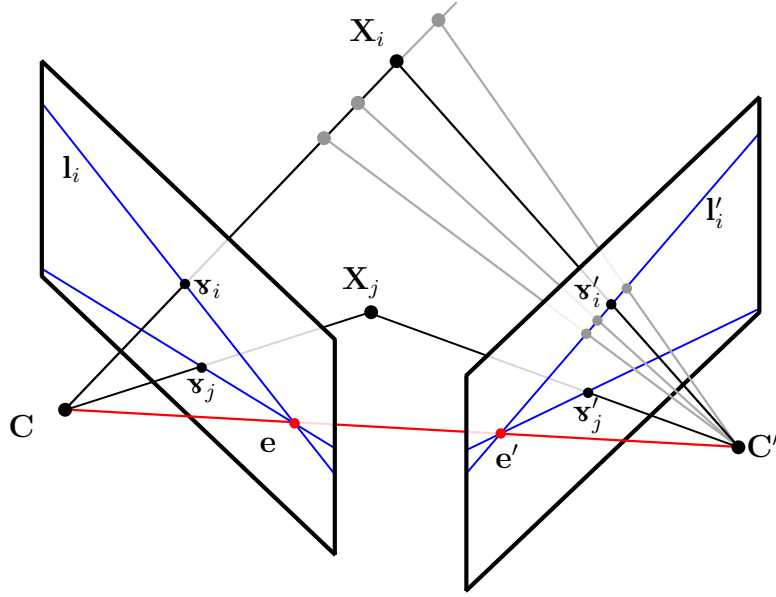


Figure 3.3: The epipolar geometry of stereo images. We have the epipoles \mathbf{e} and \mathbf{e}' as projections of the camera centres. For a projected point \mathbf{x}_i , the matching point \mathbf{x}'_i must lie on the epipolar line $\mathbf{l}'_i = E \tilde{\mathbf{x}}_i$, which is a line that passes through the epipole \mathbf{e}' . An additional example is provided as the projections of \mathbf{X}_j .

The essential matrix is a 3×3 matrix that holds the following relationship for stereo images [9]: for any point \mathbf{x} on the projection plane of P , we have the epipolar line $\tilde{\mathbf{l}}$ as the line that contains all geometrically possible coordinates for the corresponding point \mathbf{x}' given by

$$\tilde{\mathbf{l}} = E \tilde{\mathbf{x}}. \quad (3.23)$$

It can be seen in Figure 3.3 that any epipolar line will contain the epipole. Similarly we may calculate the epipolar line corresponding to a point $\tilde{\mathbf{x}}'$ on the projection plane of P' as

$$\tilde{\mathbf{l}} = E^T \tilde{\mathbf{x}}'. \quad (3.24)$$

From Equation 3.6 we have the strict equality

$$\tilde{\mathbf{x}}'^T \tilde{\mathbf{l}} = 0, \quad (3.25)$$

which leads to the epipolar constraint:

$$\tilde{\mathbf{x}}'^T E \tilde{\mathbf{x}} = 0. \quad (3.26)$$

This constraint is the basis for estimating the essential matrix from feature matches (Section 3.6), which will then be instrumental in the extraction of camera poses.

The epipoles can be calculated directly from E . Considering Equation 3.23 and Figure 3.3, \mathbf{e} has only one possible corresponding point \mathbf{e}' , which means the

line $\tilde{\mathbf{l}}' = E\tilde{\mathbf{e}}$ cannot be expressed with Euclidean geometry. In homogeneous coordinates we have

$$E\tilde{\mathbf{e}} = \mathbf{0}, \quad (3.27)$$

and similarly

$$E^T \tilde{\mathbf{e}}' = \mathbf{0}. \quad (3.28)$$

These constraints allow us to calculate the epipoles directly from the essential matrix, by obtaining its null space and left null space.

3.6 Calculating the essential matrix

The essential matrix can be calculated from normalised feature matches $\{\mathbf{x}_1, \mathbf{x}'_1\}$, $\{\mathbf{x}_2, \mathbf{x}'_2\}, \dots, \{\mathbf{x}_n, \mathbf{x}'_n\}$. Writing the epipolar constraint (Equation 3.26) for matching points $\mathbf{x}_i = [x_i \ y_i]^T$ and $\mathbf{x}'_i = [x'_i \ y'_i]^T$, we have

$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0, \quad (3.29)$$

which leads to the linear equation

$$\begin{bmatrix} x'_i x_i & x'_i y_i & x'_i & y'_i x_i & y'_i y_i & y'_i & x_i & y_i & 1 \end{bmatrix} \mathbf{E} = 0, \quad (3.30)$$

with the unknown elements of E packed into the vector $\mathbf{E} = [e_{11} \ e_{12} \ \dots \ e_{33}]^T$.

We extend this equation with multiple matches $\{\mathbf{x}_1, \mathbf{x}'_1\}, \dots, \{\mathbf{x}_n, \mathbf{x}'_n\}$ to obtain a system

$$\begin{bmatrix} x'_1 x_1 & x'_1 y_1 & x'_1 & y'_1 x_1 & y'_1 y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_n x_n & x'_n y_n & x'_n & y'_n x_n & y'_n y_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{E} = A\mathbf{E} = \mathbf{0}. \quad (3.31)$$

We find the vector \mathbf{E} , i.e. the elements of E , as a basis for the null space of A . Note that any scalar multiplication of an essential matrix renders the same essential matrix. Therefore, the corresponding vector \mathbf{E} occupies a 1D vector space and thus we can find \mathbf{E} as a basis for the null space of A .

The essential matrix has 5 degrees of freedom and can be calculated with a minimum of 5 feature matches. However, a solution with only 5 matches is not easy to formulate and requires nonlinear techniques [9]. We will therefore calculate E using a minimum of 8 matches.

The 8-point algorithm also enforces the following property: the first two singular values of E must be equal, while the third singular value must be zero [9].

A summary of the 8-point algorithm with this enforcement is as follows:

1. $A :=$ a matrix populated according to Equation 3.31,
2. $\mathbf{E} :=$ basis for $\text{null}(A)$,
3. $E :=$ a rearrangement of \mathbf{E} according to Equation 3.30,
4. $U, \Sigma, V^T := \text{SVD}(E)$,
(with $\Sigma = \text{diag}(\sigma_1, \sigma_2, \sigma_3)$)
5. $\hat{\Sigma} := \text{diag}(1, 1, 0)$,
6. $E := U\hat{\Sigma}V^T$.

Following our discussion in Section 2.4, RANSAC can be used to discard outliers from a dataset. For our application we need to calculate the essential matrix from feature matches, and it is expected that some of those matches might be incorrect. To make use of RANSAC for this purpose, we need an error metric to test a feature match $\{\mathbf{x}, \mathbf{x}'\}$ against an essential matrix E . We use the Sampson error [9]. Given an essential matrix E and a feature match $\{\mathbf{x}, \mathbf{x}'\}$, we may calculate $\{\mathbf{x} + \mathbf{r}, \mathbf{x}' + \mathbf{r}'\}$ as the closest matching points that satisfy the epipolar constraint $(\tilde{\mathbf{x}}' + \tilde{\mathbf{r}}')^T E (\tilde{\mathbf{x}} + \tilde{\mathbf{r}}) = 0$, i.e. the closest ‘true’ match for $\{\mathbf{x}, \mathbf{x}'\}$. The geometric error of the given match will be the squared distance $|\mathbf{r}|^2 + |\mathbf{r}'|^2$.

The Sampson error, as an approximation to the geometric error, is defined as

$$e = \frac{(\tilde{\mathbf{x}}'^T E \tilde{\mathbf{x}})^2}{(E \tilde{\mathbf{x}})_1^2 + (E \tilde{\mathbf{x}})_2^2 + (E^T \tilde{\mathbf{x}}')_1^2 + (E^T \tilde{\mathbf{x}}')_2^2}, \quad (3.32)$$

where $(E \tilde{\mathbf{x}})_i^2$ represents the square of the i^{th} element of $E \tilde{\mathbf{x}}$.

When calculating the essential matrix E from feature matches, we use RANSAC (Algorithm 1, Section 2.4) with the 8-point algorithm and set the input parameters as follows:

1. $\mathcal{S} :=$ a set of feature matches $\{\{\mathbf{x}_1, \mathbf{x}'_1\}, \dots, \{\mathbf{x}_n, \mathbf{x}'_n\}\}$,
2. $f(\mathcal{X}) :=$ the 8-point algorithm (with \mathcal{X} a minimum of 8 feature matches, and with output as the resulting essential matrix E),
3. $e(\mathbf{x}, M) :=$ the Sampson error (with \mathbf{x} as a feature match $\{\mathbf{x}_i, \mathbf{x}'_i\}$, M as an essential matrix E , and the output as the resulting Sampson error).

This procedure allows us to simultaneously discard incorrect feature correspondences (that violate the epipolar constraint) and determine an essential matrix. The next section describes a way of extracting camera pose information from a given essential matrix.

3.7 Estimating a camera pair

For a stereo image pair we can find camera matrices that fit the epipolar geometry. A single essential matrix E can describe the epipolar geometry of an infinite number of camera pair solutions. Therefore, in order to specify a camera pair from E , we limit the possible solutions to

$$\tilde{P} = \begin{bmatrix} I & | & \mathbf{0} \end{bmatrix} \text{ and } \tilde{P}' = R' \begin{bmatrix} I & | & -\mathbf{C}' \end{bmatrix}. \quad (3.33)$$

Next we look at the relationship between the camera centres and epipolar geometry as described in equations 3.21 and 3.22:

$$\tilde{\mathbf{e}} = \tilde{P}\tilde{\mathbf{C}}' = \begin{bmatrix} I & | & \mathbf{0} \end{bmatrix} \tilde{\mathbf{C}}' = \mathbf{C}', \quad (3.34)$$

$$\tilde{\mathbf{e}}' = \tilde{P}'\tilde{\mathbf{C}} = \begin{bmatrix} R' & | & -R'\mathbf{C}' \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = R'\mathbf{C}'. \quad (3.35)$$

Since these expressions are actually homogeneous equivalence relations, there can be a difference in scale between $\tilde{\mathbf{e}}$ and \mathbf{C}' . This ambiguity is resolved with an additional choice (that essentially fixes a scale for the entire system). By choosing that $\|\mathbf{C}'\| = 1$ and by implication, $\|R'\mathbf{C}'\| = 1$, we are left with the following possible combinations for R' and \mathbf{C}' :

$$\mathbf{C}' = \pm \frac{\tilde{\mathbf{e}}}{\|\tilde{\mathbf{e}}\|} \text{ and } R'\mathbf{C}' = \pm \frac{\tilde{\mathbf{e}}'}{\|\tilde{\mathbf{e}}'\|}. \quad (3.36)$$

Furthermore, to find R' , the following method proposed by Hartley and Zisserman [9] may be used: with $E = U\Sigma V^T$ as the singular value decomposition of E , the two possible solutions for R' are

$$U \begin{bmatrix} -\mathbf{v}_2 & \mathbf{v}_1 & \mathbf{v}_3 \end{bmatrix}^T \text{ or } U \begin{bmatrix} \mathbf{v}_2 & -\mathbf{v}_1 & \mathbf{v}_3 \end{bmatrix}^T, \quad (3.37)$$

where \mathbf{v}_i is the i^{th} column of V . Finally, \tilde{P}' can then be one of the following four possibilities [9]:

$$\begin{aligned} \tilde{P}' &= U \begin{bmatrix} -\mathbf{v}_2 & \mathbf{v}_1 & \mathbf{v}_3 \end{bmatrix}^T \begin{bmatrix} I & | & \frac{\tilde{\mathbf{e}}}{\|\tilde{\mathbf{e}}\|} \end{bmatrix} && \text{or} \\ \tilde{P}' &= U \begin{bmatrix} -\mathbf{v}_2 & \mathbf{v}_1 & \mathbf{v}_3 \end{bmatrix}^T \begin{bmatrix} I & | & \frac{-\tilde{\mathbf{e}}}{\|\tilde{\mathbf{e}}\|} \end{bmatrix} && \text{or} \\ \tilde{P}' &= U \begin{bmatrix} \mathbf{v}_2 & -\mathbf{v}_1 & \mathbf{v}_3 \end{bmatrix}^T \begin{bmatrix} I & | & \frac{\tilde{\mathbf{e}}}{\|\tilde{\mathbf{e}}\|} \end{bmatrix} && \text{or} \\ \tilde{P}' &= U \begin{bmatrix} \mathbf{v}_2 & -\mathbf{v}_1 & \mathbf{v}_3 \end{bmatrix}^T \begin{bmatrix} I & | & \frac{-\tilde{\mathbf{e}}}{\|\tilde{\mathbf{e}}\|} \end{bmatrix}, \end{aligned} \quad (3.38)$$

Figure 3.4 provides a visual guide for these four combinations.

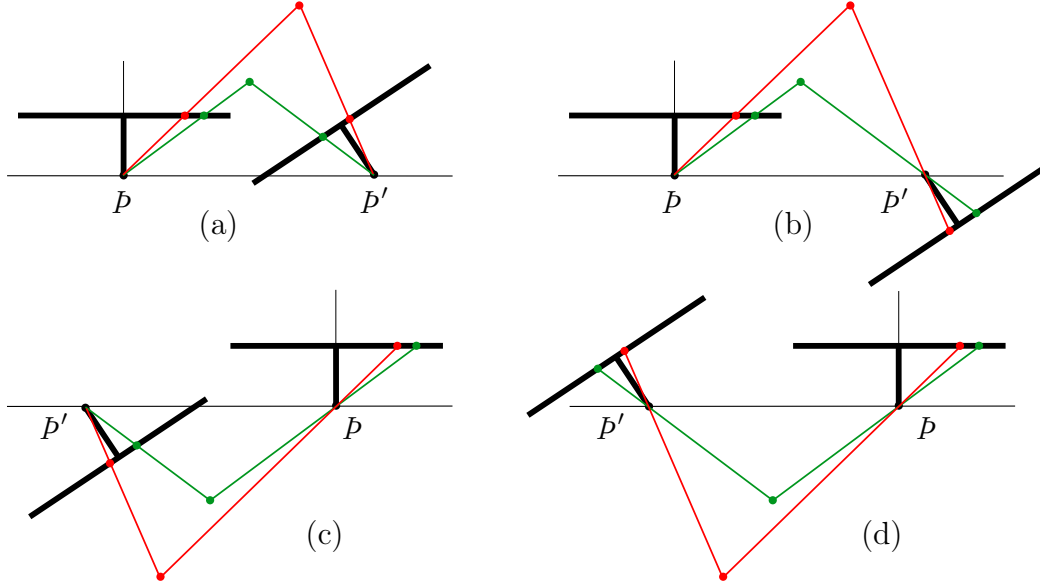


Figure 3.4: An example of the choices in Equation 3.38. Given an essential matrix E we have four possible camera pair arrangements abiding both the inherent epipolar constraints of E and the added constraints of a fixed camera P and unit distance between P and P' . Green and red represent two features and their projections onto the image planes.

It is safe to assume in practice that features correspond to 3D points that are in front of both cameras. We therefore identify the correct configuration from the possibilities in Equation 3.38 by triangulating the feature matches using all four possibilities and then determining which of the four yields points in front of both cameras.

3.8 Extension to multiple cameras

We now discuss a method to extend the pairwise camera solution given in Section 3.7 to the case of multiple cameras. There is a scale ambiguity for each of these camera pairs. Therefore, we cannot simply position the cameras according to overlapping camera pairs. We combine two overlapping camera pairs by first finding features matches across the associated three images and then combining the pairs in such a way to ensure consistent triangulation and projection between the features matches. Figure 3.5 provides an illustrative guide.

Suppose we want to add a new camera pose $P^{(n)}$ to an existing sequence

$$\mathcal{S}_a = \{P_a, P'_a, P''_a, \dots, P_a^{(n-1)}\}. \quad (3.39)$$

First we find a new camera pair from image $I^{(n)}$ and image $I^{(n-1)}$ (so as to overlap with \mathcal{S}_a) using the approach in Section 3.7. The result is

$$\mathcal{S}_b = \{P_b^{(n-1)}, P_b^{(n)}\}, \text{ with } P_b^{(n-1)} = [I \mid \mathbf{0}]. \quad (3.40)$$

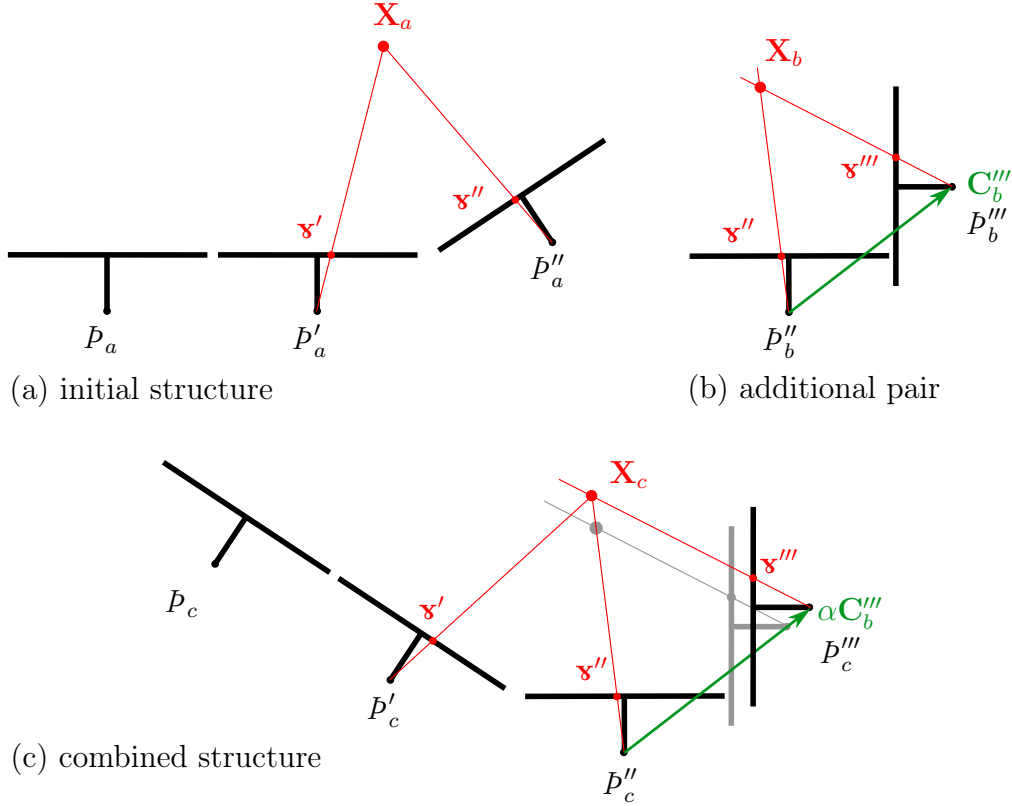


Figure 3.5: As an example we have images $\{I, I', I'', I'''\}$, matching features $\{\mathbf{x}', \mathbf{x}'', \mathbf{x}'''\}$ in images I', I'' and I''' , an initial structure (a) corresponding to $\{I, I', I''\}$, and a camera pair (b) corresponding to $\{I'', I'''\}$ to be added to the structure. By combining and rescaling the structures in (a) and (b) such as is shown in this figure, we find poses for all cameras that triangulate matching points consistently.

Next we can find a combined set \mathcal{S}_c by first adding camera set \mathcal{S}_a as

$$\mathcal{S}_c := \text{shift and rotate } \mathcal{S}_a \text{ such that } P_a^{(n-1)} \rightarrow [I | \mathbf{0}], \quad (3.41)$$

and then adding $P_c^{(n)}$, a scaled version of $P_b^{(n)}$, such that

$$P_c^{(n)} = R_b^{(n)} \left[I \mid -\alpha \mathbf{C}_b^{(n)} \right], \quad (3.42)$$

where α is a scaling factor chosen in such a manner that the triangulations between shared features are consistent (as is illustrated in Figure 3.5).

A practical and more computationally efficient approach to building \mathcal{S}_c is to keep the larger structure \mathcal{S}_a constant and rather shift, rotate and scale \mathcal{S}_b . We have chosen to show the workings of the former approach, because the calculations are easier to express. There is, however, a direct transformation between these two approaches.

To find α , we need a matching feature spanning across three images such as $\mathbf{x}^{(n-2)}$, $\mathbf{x}^{(n-1)}$ and $\mathbf{x}^{(n)}$. We then triangulate a point \mathbf{X}_c from $\mathbf{x}^{(n-2)}$ and $\mathbf{x}^{(n-1)}$

and ensure that the projection

$$\tilde{\mathbf{f}}^{(n)} = p_c^{(n)} \tilde{\mathbf{X}}_c \quad (3.43)$$

is consistent. We calculate α from the homogeneous equality in Equation 3.43 by using the strict equality

$$\tilde{\mathbf{f}}^{(n)} \times P_c^{(n)} \tilde{\mathbf{X}}_c = \mathbf{0},$$

and expand it as

$$\begin{aligned} \tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \left[I - \alpha \mathbf{C}_b^{(n)} \right] \tilde{\mathbf{X}}_c &= \mathbf{0} \\ \tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} (\lambda \mathbf{X}_c - \lambda \alpha \mathbf{C}_b^{(n)}) &= \mathbf{0} \\ \lambda \tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{X}_c - \lambda \alpha \tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{C}_b^{(n)} &= \mathbf{0} \\ \lambda \alpha \tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{C}_b^{(n)} &= \lambda \tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{X}_c \\ \alpha \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} &= (\tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{X}_c) \oslash (\tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{C}_b^{(n)}), \end{aligned} \quad (3.44)$$

where \oslash indicates element-wise division. In the case of noisy parameters, we have

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = (\tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{X}_c) \oslash (\tilde{\mathbf{f}}^{(n)} \times R_b^{(n)} \mathbf{C}_b^{(n)}), \quad (3.45)$$

where α_1 , α_2 and α_3 are not necessarily equal. In practice we estimate α by finding all candidates using all the matches and, to eliminate the influence of outliers, taking the median of these elements.

3.9 Summary

By using the theory discussed in this chapter, we may construct the following structure and motion pipeline that operates on a set of images:

1. find the calibration parameters for every camera in use,
2. find pairwise feature matches between all images with overlapping views,
3. use RANSAC to eliminate incorrect feature matches and estimate an essential matrix for every camera pair,
4. calculate camera matrix pairs from essential matrices,
5. initialise a set of camera poses with an arbitrary camera pair,
6. expand the set by adding camera pairs that overlap with cameras already within the set,
7. finally triangulate all the feature matches, using the set of camera poses, to obtain a sparse 3D model.

Figure 3.6 provides an example, where we use this pipeline with the images shown to obtain a set of camera poses and 3D model. Note that this is not a reliable method to obtain a structure and motion model. Pose estimation obtained in a pairwise manner is prone to errors like drift, where the accumulative pose error in the set propagates to newly added camera poses.

The main aim of this study is to consider the structure and motion problem in a probabilistic framework, and to refine a given set of camera poses and 3D model (such as those obtained with the above pairwise approach) through the optimisation of joint likelihoods. For that we first need some basic principles from probability theory.

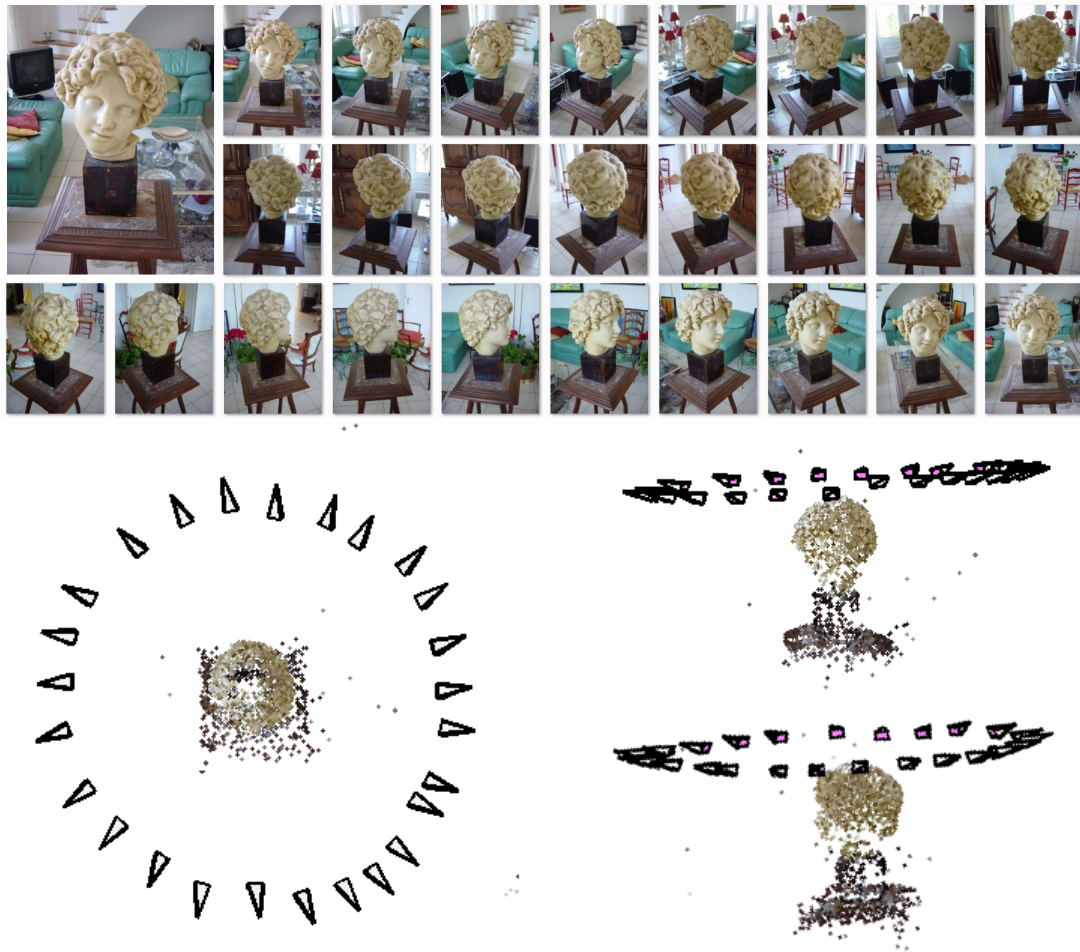


Figure 3.6: An example of estimating a 3D scene from images using the structure and motion pipeline from Section 3.9. We extracted and matched SIFT features between the images and used multiview geometry to estimate the camera poses of the images and 3D positions of the matches. The image set is taken from [43].

Chapter 4

Basic concepts in probability

To be able to formulate the structure and motion problem probabilistically, we first need to express multiple view geometry by using probability distributions. This chapter serves as the basis for expressing the parameters of a system, along with the transformations within the system, probabilistically. The next chapter will focus on combining such parameters into a graph structure and finding the most probable solution for all the parameters involved. This will enable us to construct and solve our own probabilistic model of the structure and motion problem.

4.1 Probability distributions

A probability distribution is a representation of the uncertainty inherent in random variables. For consistency, we use capital letters to denote random variables, such as X and Y , and bold capital letters to denote random vectors, such as \mathbf{X} and \mathbf{Y} . We will mostly express a random variable in terms of its probability density function (pdf) or, in the case of a discrete random variable, its probability mass function. The random variable X with random distribution $P(X)$ can be expressed by the probability density (or mass) function $p_X(x)$. When it is clear from context we may drop the subscript and denote the function as $p(x)$.

The discussion will now focus on the following types of distributions: joint probability distributions, marginal probability distributions and conditional probability distributions. Consult Figure 4.1 as a visual guide to this discussion.

1. A joint distribution is a distribution that describes the combined probability of two or more random variables. For instance, given the random variables X_1, X_2, \dots, X_n , their joint distribution $P(X_1, X_2, \dots, X_n)$ can be represented by the multivariate function $p_{X_1, \dots, X_n}(x_1, \dots, x_n)$.
2. A marginal distribution refers to a distribution of a subset of random variables with regards to a joint distribution, that does not depend on information about the rest of the joint's variables. We can find such a

marginal distribution by integrating over the unwanted variables. For example, to find the marginal distribution over X_3, \dots, X_n when given $P(X_1, \dots, X_n)$, we integrate over X_2 and X_1 :

$$p_{X_3, \dots, X_n}(x_3, \dots, x_n) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p_{X_1, \dots, X_n}(x_1, \dots, x_n) dx_2 dx_1. \quad (4.1)$$

3. A conditional distribution, with regards to a joint distribution, is a distribution where some of the random variables are observed (i.e the variables are known to be particular values) and the other random variables are unobserved (i.e. the outcomes of those variables remain uncertain). The conditional distribution

$$p_{X_3, \dots, X_n | X_1, X_2}(x_3, \dots, x_n | X_1 = x_1, X_2 = x_2), \quad (4.2)$$

where x_1 and x_2 are observations of X_1 and X_2 respectively, is equal to

$$p_{X_1, \dots, X_n}(x_1, \dots, x_n) |_{X_1 = x_1 \text{ and } X_2 = x_2}.$$

As previously mentioned, we often drop the subscript when using this notation. Additionally, if no ambiguity is introduced, we often reduce the writing out of random variable assignments $X=x$ to x . For example

$$p(x_3, \dots, x_n | x_1, x_2) = p_{X_3, \dots, X_n | X_1, X_2}(x_3, \dots, x_n | X_1 = x_1, X_2 = x_2).$$

Furthermore, by introducing a few rules, a probability distribution can be broken up into factors. If there are some statistical independencies between random variables within a joint distribution, the distribution can be factorised in such a way that the largest factor is of a lower dimensionality than the full joint distribution. This can be useful for finding a lower-dimensional equivalent of a high-dimensional joint distribution.

For two random variables X and Y the **product rule**, as illustrated in Figure 4.2, states that

$$P(X, Y) = P(X|Y) P(Y), \quad (4.3)$$

which can be extended to the **chain rule**, when presented with three or more variables:

$$P(X_1, \dots, X_n) = \prod_{i \in \{1, \dots, n\}} P(X_i | X_{i+1}, \dots, X_n). \quad (4.4)$$

If two variables X and Y are **independent**, which we denote as $X \perp\!\!\!\perp Y$, information about Y will not affect the probability distribution of X :

$$P(X, Y) = P(X) P(Y). \quad (4.5)$$

In this case the 2D pdf $p_{X,Y}(x, y)$ can be represented by the two 1D pdfs $p_X(X)$ and $p_Y(Y)$. Also, by combining Equation 4.3 and Equation 4.5, we note that the conditional pdf $p_{X|Y}(x|y)$ can be represented by $p_X(x)$.

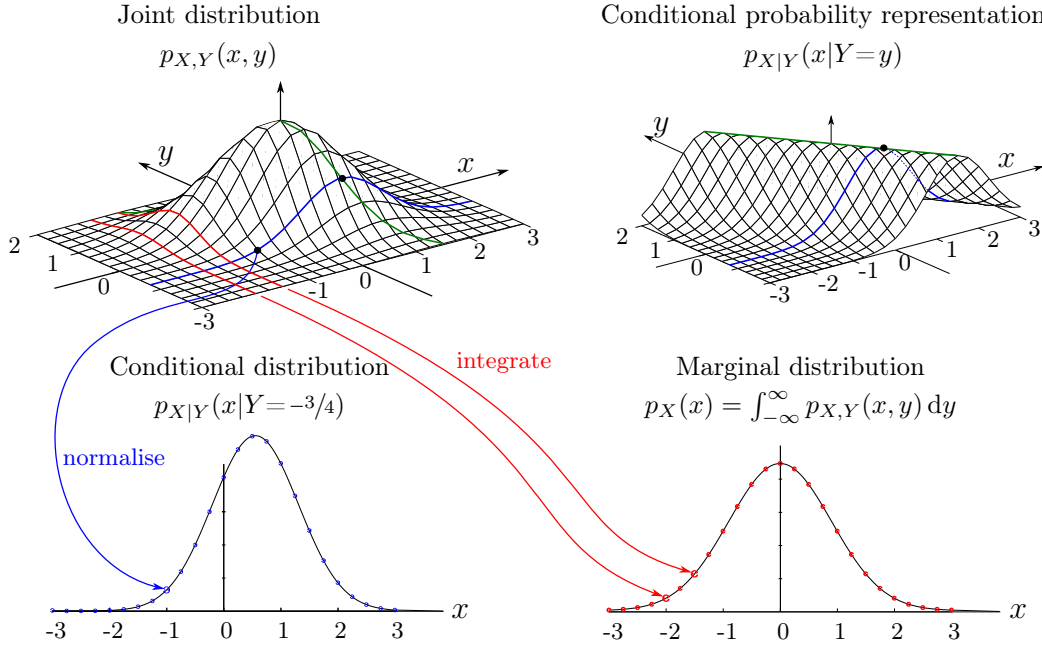


Figure 4.1: An example of a joint, conditional and marginal probability distribution. **Joint:** $P(X, Y)$ (top left) is the joint distribution over two random variables X and Y . **Conditional:** The general conditional distribution $P(X|Y)$ (top right) is presented by plotting the pdf over all possible observations of Y . The conditional $P(X|Y=-3/4)$ (bottom left) is presented as a cut from the joint distribution where $Y=-3/4$. **Marginal:** The marginal distribution $P(X)$ (bottom right) is obtained by integrating the distribution $P(X, Y)$ over Y .

Furthermore, two variables X and Y are **conditionally independent** with regards to a third variable Z , expressed as $X \perp\!\!\!\perp Y|Z$, if and only if, by conditioning on Z , X and Y become independent:

$$P(X, Y|Z) = P(X|Z) P(Y|Z). \quad (4.6)$$

If the independencies and conditional independencies between the random variables are known, the factors within the chain rule can be reduced.

4.2 Discrete probability tables

In order to implement a system using probabilistic models, we need to have structures that can be stored and manipulated efficiently. We introduce discrete probability tables, an array structure representing discrete probabilistic space, and define some operations associated with these discrete probability tables.

We can store the values of a given discrete probability mass function

$$p_{X_1, \dots, X_n}(x_1, \dots, x_n)$$

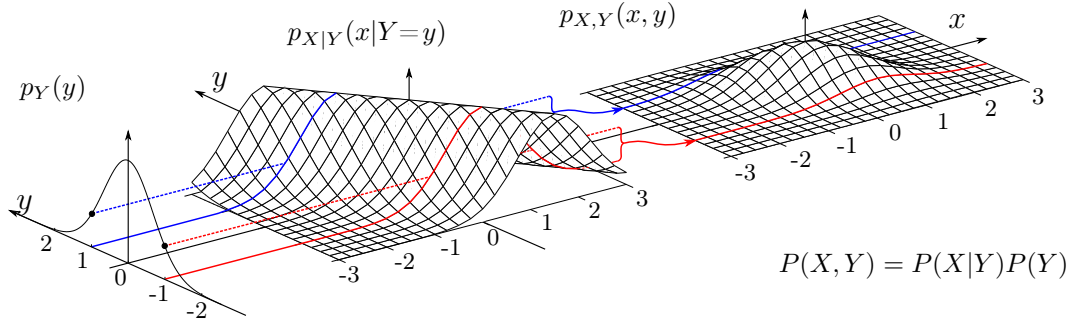


Figure 4.2: An example of the product rule for finding the joint distribution $P(X, Y)$ from the marginal $P(Y)$ and the conditional $P(X|Y)$.

in an n -dimensional array structure, labelled according to the random variables. We can also use such an array structure in the case of conditional distributions. Given the probability mass function for a conditional probability with n unknown variables and m observed variables as

$$p(x_1, \dots, x_n | X_{n+1}=x_{n+1}, \dots, X_{n+m}=x_{n+m}),$$

where x_{n+1}, \dots, x_{n+m} are constant values, we will be left with an n -dimensional array. However, if we need x_{n+1}, \dots, x_{n+m} as variables, we can represent all possible distributions for all possible observations in an $(n + m)$ -dimensional array. An example of some probability tables can be found in Figure 4.3.

$P(X, Y, Z)$				$P(X)$		$P(Y Z=z)$	
	y_1	y_2				z_1	z_2
x_1	0.0162	0.0108	0.1485	0.27	0.600	0.611	
			0.2090				
	0.0228	0.0152	0.1925	0.38	0.400	0.389	
x_2			0.1225				
x_3	0.0210	0.0140		0.35			
			z_1				z_2

Figure 4.3: An example of a joint probability table for $P(X, Y, Z)$, a marginal table for $P(X)$ and a conditional table for $P(Y|Z=z)$.

Note that there are cases when these discrete probability tables contain redundant information. Given $P(X, Y)$, if we want to represent $P(X|Y)$ as a discrete probability table, we will have a 2-dimensional array. If we are further given the information that $X \perp Y$, we have $P(X|Y) = P(X)$, which is a 1-dimensional array. In this case $P(X|Y)$ can be expressed by a 1-dimensional array.

We will now show how to apply conditioning, marginalisation, division and multiplication on these tables. Let us consider a joint probability distribution $P(X, Y, Z)$ (as seen in Figure 4.3) with $X \perp \{Y, Z\}$ and, therefore,

$$P(X, Y, Z) = P(X)P(Y|Z)P(Z).$$

Multiplication: For our first example, we show the product $P(X, Y, Z) = P(X)P(Y, Z)$, and for our second example, we show how multiplication is applied with regards to a conditional table with $P(Y, Z) = P(Y|Z=z)P(Z)$. The procedure, as shown in Figure 4.4 and 4.5, is quite simple: we create a new probability table by multiplying each combination of states where common variables overlap.

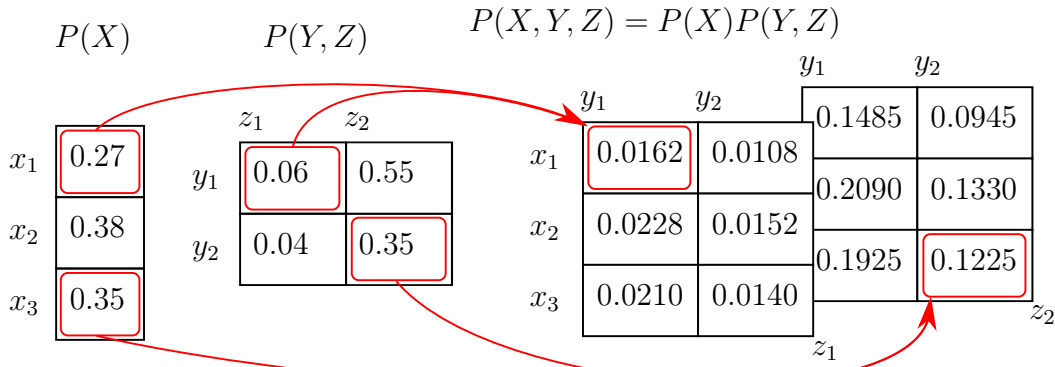


Figure 4.4: An examples of multiplying two independent distributions. Note that the dimension of the resulting distribution is $\dim(X) + \dim(\{Y, Z\})$.

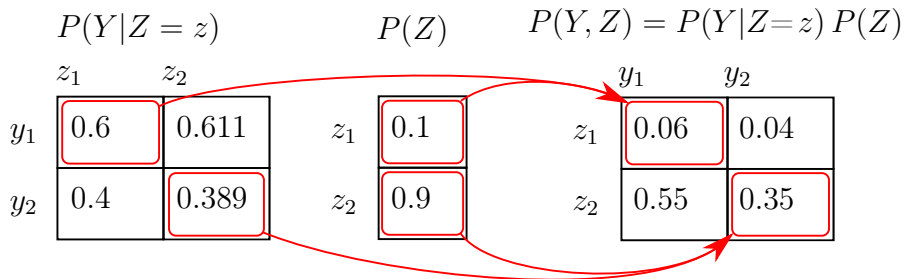


Figure 4.5: An example of multiplying two probability tables sharing a common space, with one of the tables as a conditional table.

Division: As an example of division we find the following conditional distribution:

$$P(Y|Z=z) = \frac{P(Y, Z)}{P(Z)},$$

as shown in Figure 4.6. Such division is done component-wise between matching variables, with the component division $\frac{0}{0}$ defined as 0.

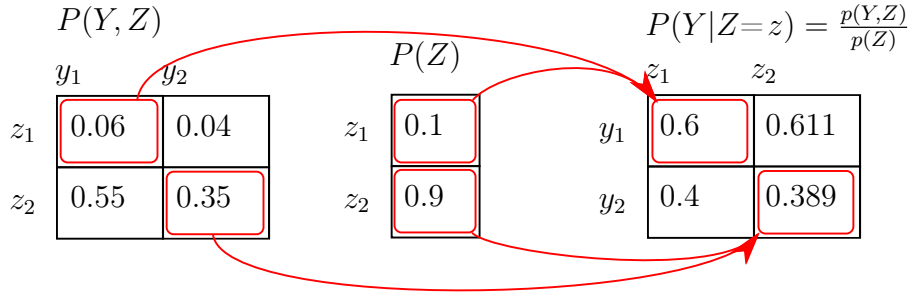


Figure 4.6: An example of division with probability tables. Note that if we had $Y \perp Z$, the resulting probability table would have had a redundant dimension.

Marginalisation: As an example we obtain the marginal $P(Y, Z)$ from the joint $P(X, Y, Z)$ by summing over all the states that are not present in the marginal. In this case $P(Y, Z) = \sum_{X=x} P(X, Y, Z)$, as shown in Figure 4.7.

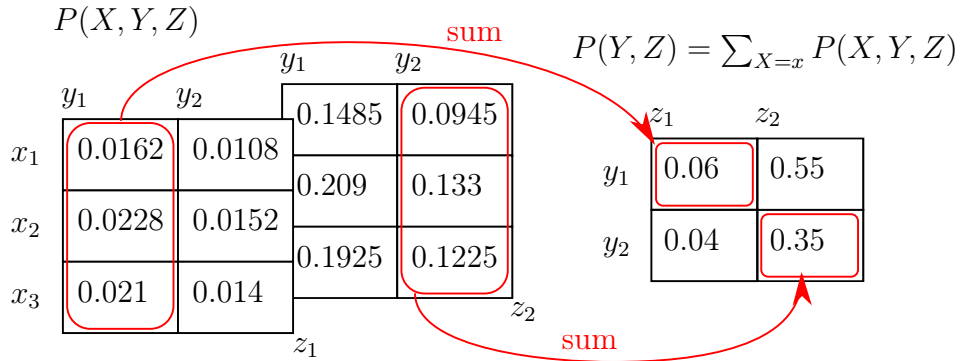


Figure 4.7: An example of marginalising a probability table.

Conditioning: In the case where a random variable is observed to be in a specific state, the resulting probability table is reduced to the dimensions of the unknown variables. For example, we determine $P(X, Y|X=x_1)$ given the joint distribution $P(X, Y, Z)$ as shown in Figure 4.8.

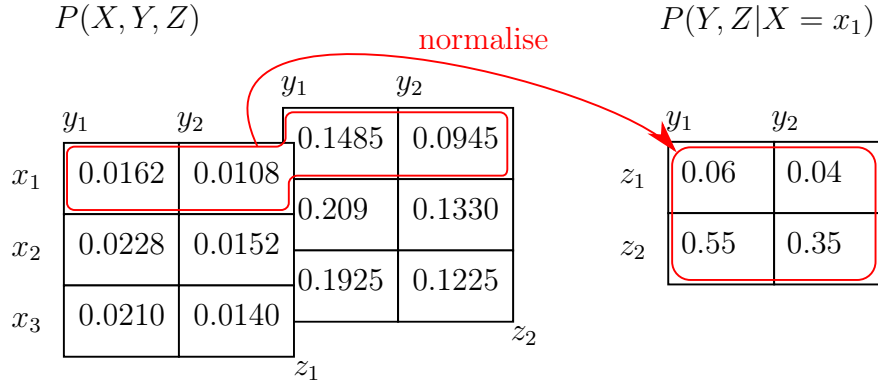


Figure 4.8: An example of finding a conditional probability table by observing the state of X .

We also have the general case where all possible conditionals are stored in a single table. For example the conditional distribution $P(Y, Z | X = x)$, where x can take on any state, is derived from the joint distribution $P(X, Y, Z)$ as expressed in Figure 4.9. Note that in this specific example $X \perp\!\!\!\perp \{Y, Z\}$ and therefore each observed state in X yields the same distribution over Y and Z .

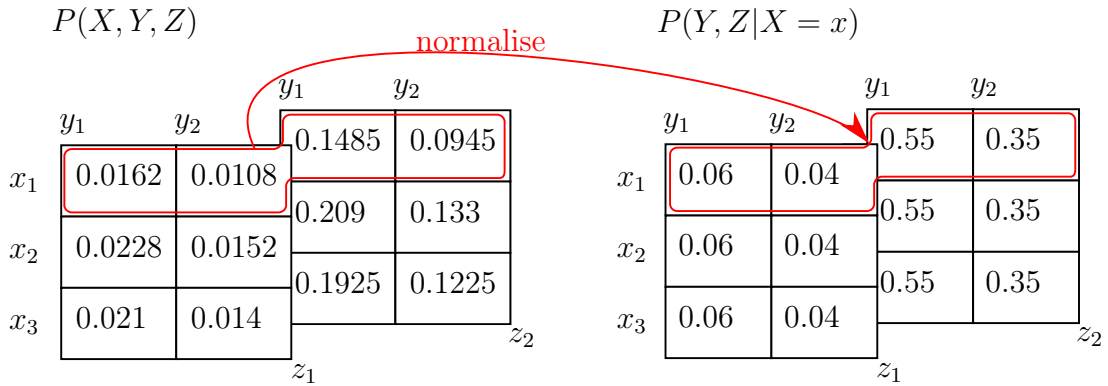


Figure 4.9: An example of a conditional table containing a random distributions for all possible conditioned states.

Factors: The probability tables we presented thus far abide by the restrictions of discrete probability distributions, such as the sum of all the elements of a probability distribution is equal to 1 and every element of the array is a number between 0 and 1. We refer to arrays that are not necessarily normalised as factors. We often use factors in our calculations where scale can safely be ignored and recovered afterwards through normalisation.

As an example, the product of probability tables

$$P(X, Y, Z) = P(X)P(Y|X)P(Z)$$

is equivalent to the product of factors of arbitrary scale

$$\psi(X, Y, Z) = \phi_1(X)\phi_2(Y, Z)\phi_3(Z),$$

since $\psi(X, Y, Z)$ can be normalised as follows (no information is lost due to the different scales):

$$P(X, Y, Z) = \frac{1}{k}\psi(X, Y, Z), \text{ where}$$

$$k = \sum \psi(X, Y, Z).$$

4.3 Multivariate Gaussian distributions

We will now focus on a Gaussian analogue to the probability tables, to be used with Gaussian random variables. We introduce the multivariate Gaussian distribution along with the necessary operations for it to be implemented in code.

A multivariate random variable (also referred to as a random vector), is a single representation of a set of random variables. A multivariate Gaussian variable, such as \mathbf{X} , can be parameterised by its mean vector $\boldsymbol{\mu}_{\mathbf{X}}$ and covariance matrix $\Sigma_{\mathbf{X}}$ with the following notation:

$$\mathcal{N}(\boldsymbol{\mu}_{\mathbf{X}}, \Sigma_{\mathbf{X}}). \quad (4.7)$$

The probability density function of this multivariate Gaussian variable is

$$p_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{|2\pi\Sigma_{\mathbf{X}}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_{\mathbf{X}})^T \Sigma_{\mathbf{X}}^{-1}(\mathbf{x} - \boldsymbol{\mu}_{\mathbf{X}})\right). \quad (4.8)$$

Note that the covariance matrix needs to be symmetric and positive-definite for the distribution to be nondegenerate.

To introduce the operations for conditioning, marginalisation, division and multiplication, we make use of the following interchangeable forms:

$$\text{covariance form: } p(\mathbf{X}; \boldsymbol{\mu}, \Sigma) = \mathcal{N}(\boldsymbol{\mu}, \Sigma), \quad (4.9)$$

$$\text{canonical form: } \mathcal{C}(\mathbf{X}; K, \mathbf{h}) = \mathcal{N}(\boldsymbol{\mu}, \Sigma), \quad (4.10)$$

with $\Sigma = K^{-1}$ and $\boldsymbol{\mu} = \Sigma\mathbf{h}$. Further discussion and proofs for the results given below can be found in Koller [44].

Multiplication: Given random Gaussian vectors \mathbf{X} and \mathbf{Y} , we can find the product of the two distributions $P(\mathbf{Z}) = P(\mathbf{X})P(\mathbf{Y})$ as:

$$\mathcal{C}(\mathbf{X}; K_{\mathbf{X}}, \mathbf{h}_{\mathbf{X}})\mathcal{C}(\mathbf{Y}; K_{\mathbf{Y}}, \mathbf{h}_{\mathbf{Y}}) = \mathcal{C}(\mathbf{Z}; K'_{\mathbf{X}} + K'_{\mathbf{Y}}, \mathbf{h}'_{\mathbf{X}} + \mathbf{h}'_{\mathbf{Y}}). \quad (4.11)$$

This calculation can only be accomplished if the scope of all the canonical forms are extended to a common one. Such a common scope could be accomplished

as follows:

$$K'_{\mathbf{X}} = \begin{bmatrix} K_{\mathbf{X}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{h}'_{\mathbf{X}} = \begin{bmatrix} \mathbf{h}_{\mathbf{X}} \\ \mathbf{0} \end{bmatrix}, \quad (4.12)$$

$$K'_{\mathbf{Y}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & K_{\mathbf{Y}} \end{bmatrix}, \quad \mathbf{h}'_{\mathbf{Y}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{h}_{\mathbf{Y}} \end{bmatrix}. \quad (4.13)$$

We might have to take conflicting information, such as prior information and new evidence, into account regarding the same event or measurement. For example, say we produced two distributions $\hat{p}_{X,1}(x)$ and $\hat{p}_{X,2}(x)$ by taking two sets of measurements of the same event. They would therefore be two approximations of the pdf of X , occupying the same probabilistic space. If we want a distribution $\hat{p}_X(x)$ by combining $\hat{p}_{X,1}(x)$ and $\hat{p}_{X,2}(x)$, we end up with a new estimation occupying the same probabilistic space.

Similarly, if the vectors \mathbf{X} and \mathbf{Y} share random variables that occupy the same probabilistic space, we define factor multiplication as follows. Find a common scope and ensure that the placement order of the random variables within \mathbf{X}' and \mathbf{Y}' are corrected so that common indices between $K'_{\mathbf{X}}$, $K'_{\mathbf{Y}}$, and $\mathbf{h}'_{\mathbf{X}}$, $\mathbf{h}'_{\mathbf{Y}}$ refer to the same variable space, then find the distribution of \mathbf{Z} as $\mathcal{C}(\mathbf{Z}; K'_{\mathbf{X}} + K'_{\mathbf{Y}}, \mathbf{h}'_{\mathbf{X}} + \mathbf{h}'_{\mathbf{Y}})$. Note that with variable space overlap this operation does not return the product of two Gaussian distributions, thus we refer to the latter operation simply as factor multiplication.

Division: We can find a multivariate distribution of \mathbf{Z} as the distribution of \mathbf{X} divided by the distribution of \mathbf{Y} :

$$\frac{\mathcal{C}(\mathbf{X}; K_{\mathbf{X}}, \mathbf{h}_{\mathbf{X}})}{\mathcal{C}(\mathbf{Y}; K_{\mathbf{Y}}, \mathbf{h}_{\mathbf{Y}})} = \mathcal{C}(\mathbf{Z}; K'_{\mathbf{X}} - K'_{\mathbf{Y}}, \mathbf{h}'_{\mathbf{X}} - \mathbf{h}'_{\mathbf{Y}}). \quad (4.14)$$

We follow the same rules regarding scope expansion as with multiplication, although it should be noted that the random variables in the denominator must be a subset of those in the numerator.

Marginalisation: Marginalisation is easy in the covariance form. Given a random vector \mathbf{Z} representing the joint distribution $P(\mathbf{Z}) = P(\mathbf{X}, \mathbf{Y})$ with the covariance and canonical parameters as

$$\Sigma_{\mathbf{Z}} = \begin{bmatrix} \Sigma_{\mathbf{X},\mathbf{X}} & \Sigma_{\mathbf{X},\mathbf{Y}} \\ \Sigma_{\mathbf{Y},\mathbf{X}} & \Sigma_{\mathbf{Y},\mathbf{Y}} \end{bmatrix}, \quad \boldsymbol{\mu}_{\mathbf{Z}} = \begin{bmatrix} \boldsymbol{\mu}_{\mathbf{X}} \\ \boldsymbol{\mu}_{\mathbf{Y}} \end{bmatrix}, \quad (4.15)$$

$$K_{\mathbf{Z}} = \begin{bmatrix} K_{\mathbf{X},\mathbf{X}} & K_{\mathbf{X},\mathbf{Y}} \\ K_{\mathbf{Y},\mathbf{X}} & K_{\mathbf{Y},\mathbf{Y}} \end{bmatrix}, \quad \mathbf{h}_{\mathbf{Z}} = \begin{bmatrix} \mathbf{h}_{\mathbf{X}} \\ \mathbf{h}_{\mathbf{Y}} \end{bmatrix}, \quad (4.16)$$

the marginal distribution $P(\mathbf{X})$, for example, is simply

$$p(\mathbf{X}; \boldsymbol{\mu}_{\mathbf{X}}, \Sigma_{\mathbf{X},\mathbf{X}}). \quad (4.17)$$

In the case of the canonical form

$$C(\mathbf{X}; K'_{\mathbf{X}}, \mathbf{h}'_{\mathbf{X}}), \quad (4.18)$$

the parameters are not as straightforward, since $K'_{\mathbf{X}} \neq K_{\mathbf{X},\mathbf{X}}$ and $\mathbf{h}'_{\mathbf{X}} \neq \mathbf{h}_{\mathbf{X}}$.

The conversion between the covariance and canonical form can be expensive for large distributions. We therefore introduce a method to reduce the expense of marginalising when presented with the canonical parameters of $P(\mathbf{Z})$. By using the blockwise matrix inversion identity [45]

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{bmatrix}, \quad (4.19)$$

with $M = (A - BD^{-1}C)^{-1}$, we find the relationship between the canonical and covariance parameters as

$$\begin{bmatrix} K_{\mathbf{X},\mathbf{X}} & K_{\mathbf{X},\mathbf{Y}} \\ K_{\mathbf{Y},\mathbf{X}} & K_{\mathbf{Y},\mathbf{Y}} \end{bmatrix}^{-1} = \begin{bmatrix} \Sigma_{\mathbf{X},\mathbf{X}} & \Sigma_{\mathbf{X},\mathbf{Y}} \\ \Sigma_{\mathbf{Y},\mathbf{X}} & \Sigma_{\mathbf{Y},\mathbf{Y}} \end{bmatrix}, \quad (4.20)$$

with $\Sigma_{\mathbf{X},\mathbf{X}} = (K_{\mathbf{X},\mathbf{X}} - K_{\mathbf{X},\mathbf{Y}}K_{\mathbf{Y},\mathbf{Y}}^{-1}K_{\mathbf{Y},\mathbf{X}})^{-1}$.

We can therefore obtain the marginal $P(\mathbf{X})$ in the canonical form (Equation 4.18) with

$$K'_{\mathbf{X}} = K_{\mathbf{X},\mathbf{X}} - K_{\mathbf{X},\mathbf{Y}}K_{\mathbf{Y},\mathbf{Y}}^{-1}K_{\mathbf{Y},\mathbf{X}}, \quad (4.21)$$

and

$$\begin{aligned} \mathbf{h}'_{\mathbf{X}} &= \Sigma_{\mathbf{X},\mathbf{X}}^{-1} \boldsymbol{\mu}_{\mathbf{X}} \\ &= K_{\mathbf{X},\mathbf{X}} \boldsymbol{\mu}_{\mathbf{X}} - K_{\mathbf{X},\mathbf{Y}}K_{\mathbf{Y},\mathbf{Y}}^{-1}K_{\mathbf{Y},\mathbf{X}} \boldsymbol{\mu}_{\mathbf{X}} \\ &= \mathbf{h}_{\mathbf{X}} - K_{\mathbf{X},\mathbf{Y}}K_{\mathbf{Y},\mathbf{Y}}^{-1} \mathbf{h}_{\mathbf{Y}}. \end{aligned} \quad (4.22)$$

Conditioning: For a given joint distribution $P(\mathbf{X}, \mathbf{Y})$, suppose we are interested in $P(\mathbf{X}|\mathbf{Y}=\mathbf{y}_0)$. We can represent this in canonical form [44]:

$$\begin{aligned} &\mathcal{C}(\mathbf{X}; K'_{\mathbf{X}}, \mathbf{h}'_{\mathbf{X}}), \text{ with} \\ &K'_{\mathbf{X}} = K_{\mathbf{X},\mathbf{X}}, \text{ and} \\ &\mathbf{h}'_{\mathbf{X}} = \mathbf{h}_{\mathbf{X}} - K_{\mathbf{X},\mathbf{Y}} \mathbf{y}_0. \end{aligned} \quad (4.23)$$

Note that we are unable to represent a general covariance $P(\mathbf{X}|\mathbf{Y}=\mathbf{y})$, where \mathbf{y} is a variable, in canonical or covariance form.

4.4 Transformations on Gaussian distributions

When a linear transformation is applied to a multivariate Gaussian variable \mathbf{X} , such as

$$\mathbf{Y}' = A^T \mathbf{X} + \mathbf{c}, \quad (4.24)$$

we obtain the covariance parameters of the resulting random vector \mathbf{Y}' as

$$\boldsymbol{\mu}_{\mathbf{Y}'} = A^T \boldsymbol{\mu}_{\mathbf{X}} + \mathbf{c} \text{ and } \Sigma_{\mathbf{Y}', \mathbf{Y}'} = A^T \Sigma_{\mathbf{X}, \mathbf{X}} A. \quad (4.25)$$

If we represent both random vectors \mathbf{X} and \mathbf{Y}' as a single joint probability distribution $P(\mathbf{X}, \mathbf{Y}')$, we may encounter singularity problems since the one distribution determines the other. For instance, if we observe $\mathbf{X} = \mathbf{x}_0$ in an attempt to obtain $P(\mathbf{Y}' | \mathbf{X} = \mathbf{x}_0)$, \mathbf{Y}' would also be observed as $\mathbf{Y}' = A^T \mathbf{x}_0 + \mathbf{c}$.

We combat this situation by adding zero mean noise with covariance $\Sigma_{\mathbf{N}}$, which we refer to as measurement noise, to the linear transformation:

$$\mathbf{Y} = A^T \mathbf{X} + \mathbf{c} + \mathbf{N}. \quad (4.26)$$

This enables information from \mathbf{X} to not strictly determine \mathbf{Y} and, in doing so, enables the vectors to be jointly Gaussian.

We now have the marginal distribution concerning \mathbf{Y} parameterised by

$$\begin{aligned} \boldsymbol{\mu}_{\mathbf{Y}} &= A^T \boldsymbol{\mu}_{\mathbf{X}} + \mathbf{c}, \\ \Sigma_{\mathbf{Y}, \mathbf{Y}} &= A^T \Sigma_{\mathbf{X}, \mathbf{X}} A + \Sigma_{\mathbf{N}, \mathbf{N}}, \end{aligned} \quad (4.27)$$

and the joint covariance as

$$\Sigma_{\mathbf{X}, \mathbf{Y}} = \Sigma_{\mathbf{X}, \mathbf{X}} A, \quad (4.28)$$

which leads to \mathbf{Z} as the random vector for the full joint representation $P(\mathbf{X}, \mathbf{Y})$, parameterised by

$$\begin{aligned} \boldsymbol{\mu}_{\mathbf{Z}} &= \begin{bmatrix} \boldsymbol{\mu}_{\mathbf{X}} \\ A^T \boldsymbol{\mu}_{\mathbf{X}} + \mathbf{c} \end{bmatrix}, \\ \Sigma_{\mathbf{Z}} &= \begin{bmatrix} \Sigma_{\mathbf{X}, \mathbf{X}} & \Sigma_{\mathbf{X}, \mathbf{X}} A \\ A^T \Sigma_{\mathbf{X}, \mathbf{X}} & A^T \Sigma_{\mathbf{X}, \mathbf{X}} A + \Sigma_{\mathbf{N}, \mathbf{N}} \end{bmatrix}. \end{aligned} \quad (4.29)$$

4.5 Sigma point parameterisation

We need to be able to linearise nonlinear transformations of random distributions in order to fully model our structure and motion problem with multivariate Gaussian distributions. We make use of the unscented transform by Uhlmann [23] to model nonlinear projections of Gaussian mean and covariance estimates. The first advantage of this approach is that the unscented transform can be applied with any given function whereas alternative linearisation methods, such as Taylor series approximations, may not be suitable for functions that are not differentiable. Secondly there is no need to derive and implement a Jacobian matrix.

The unscented transform operates as follows. When presented with sufficiently many measurements of an unknown multivariate Gaussian distribution, the

distribution can be estimated by calculating the sample mean and the sample covariance of these measurements. It is also possible to contrive samples in such a way that the sample mean and sample covariance are equivalent to those of a given multivariate Gaussian. Uhlmann referred to these samples as sigma points. We will explore methods to choose a set of sigma points and show how they can be used in estimating Gaussian distributions resulting from nonlinear transformations.

4.5.1 General sigma point formulation

The Cholesky factorisation allows us to split a $D \times D$ positive-definite covariance matrix Σ into the product of a lower-triangular matrix L and the transpose L^T as

$$\Sigma = LL^T = \sum_{i=1}^D \mathbf{L}_i \mathbf{L}_i^T, \quad (4.30)$$

with the columns of the lower-triangular Cholesky matrix as \mathbf{L}_i .

By constructing sigma points with equal weights from the Cholesky factorisation as

$$\mathbf{s}_{i\pm} = \boldsymbol{\mu} \pm k\mathbf{L}_i, \quad \text{with } w_i = w = 1, \quad \text{for } i = 1, \dots, D, \quad (4.31)$$

where \mathbf{s}_i are the sigma points, w_i are the weights and the constant $k = \sqrt{D}$, we find that the sample mean is equal to $\boldsymbol{\mu}$ and the sample covariance matrix is also the same as the original distribution:

$$\begin{aligned} \hat{\Sigma} &= \frac{\sum_i w_i (\mathbf{s}_i - \boldsymbol{\mu})(\mathbf{s}_i - \boldsymbol{\mu})^T}{\sum_i w_i} \\ &= \frac{w \sum_i k^2 \mathbf{L}_i \mathbf{L}_i^T + w \sum_i (-k)^2 \mathbf{L}_i \mathbf{L}_i^T}{2Dw} \\ &= \frac{2wk^2 \sum_i \mathbf{L}_i \mathbf{L}_i^T}{2Dw} \\ &= \frac{k^2 \Sigma}{D} \\ &= \Sigma. \end{aligned} \quad (4.32)$$

One of the main advantages of sigma points is the relationship with linear transformations. For a multivariate Gaussian variable \mathbf{X} with sigma points \mathbf{s}_i , a linear transformation $\mathbf{Y} = A\mathbf{X} + \mathbf{c}$ with resulting sigma points \mathbf{t}_i will exhibit the same linear relationship between the sigma points: $\mathbf{t}_i = A\mathbf{s}_i + \mathbf{c}$ [23]. An example of this choice of sigma points is given in Figure 4.10.

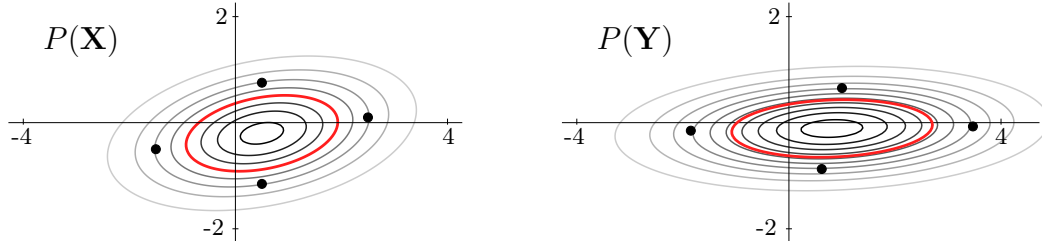


Figure 4.10: An example of a linear transformation with sigma points. On the left we have a multivariate Gaussian over \mathbf{X} (as a contour plot with one standard deviation away from the mean marked in red), and its associated sigma points \mathbf{s}_i . On the right we have \mathbf{Y} from the linear transformation $\mathbf{Y} = A\mathbf{X} + \mathbf{c}$ with its associated sigma points \mathbf{t}_i . These sigma points can alternatively be obtained through the same transformation: $\mathbf{t}_i = A\mathbf{s}_i + \mathbf{c}$.

4.5.2 Other sigma point formulations

There are other valid arrangements such as **the standard sigma point formulation**, which allows for an additional point to be placed at the mean. For nonnegative weights we can find a suitable spacing for the sigma points by choosing a weight for the centroid $w_0 < 1$, and then determining the spacing and weights of the other point as

$$\begin{aligned} \mathbf{s}_0 &= \boldsymbol{\mu} & \text{with } w_0 < 1, \\ \text{and } \mathbf{s}_{i\pm} &= \boldsymbol{\mu} \pm \sqrt{\frac{D}{1-w_0}} L_i & \text{with } w_{i\pm} = \frac{1-w_0}{2D} \quad \text{for } i = 1, \dots, D. \end{aligned} \quad (4.33)$$

If negative weights are allowed, a valid option is to place the $i \neq 0$ sigma points one standard deviation from the mean and calculate $w_0 = 1 - D$, which results in a negative w_0 . This is problematic for some applications such as clustering, where a cluster might end up with a covariance matrix that is not positive-definite.

Another alternative is to choose a positive w_0 and calculate the other parameters. The trade-off is that, for large values of D , the remaining sigma points will be placed multiple standard deviations away from the mean. This is problematic for applications such as linearisation where the focus needs to be close to the mean.

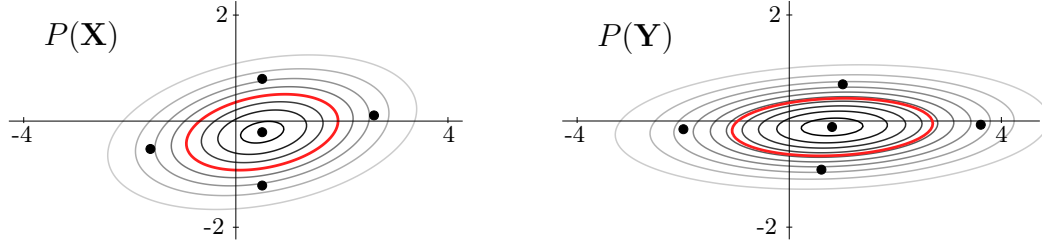


Figure 4.11: Example of the standard sigma point form with $w_0 = \frac{0.5}{2D+1} = 0.1$ and $w_1 = 0.225$. The linear transformation $\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{c}$ is the same as presented in Figure 4.10.

Yet another option is the **extended sigma point formulation** [46], where we have three sets of sigma points. We choose the first set at the mean, the second set at one standard deviation ($k = 1$), and the third set at $k = \sqrt{2D - 0.5}$, all with equal weight w :

$$\begin{aligned} \mathbf{s}_0 &= \boldsymbol{\mu}, \\ \mathbf{s}_{i\pm} &= \boldsymbol{\mu} \pm L_i && \text{for } i = 1, \dots, D, \\ \mathbf{s}_{i\pm} &= \boldsymbol{\mu} \pm \sqrt{2D - 0.5} L_i && \text{for } i = D + 1, \dots, 2D. \end{aligned} \quad (4.34)$$

All of the sigma points are positively weighted, a large portion is located close to one standard deviation, and the sample covariance is

$$\begin{aligned} \hat{\Sigma} &= \frac{w\mathbf{0}\mathbf{0}^T + 2w \sum_i L_i L_i^T + 2w \sum_i (2D - 0.5) L_i L_i^T}{w + 2wD + 2wD} \\ &= \frac{(2w + 2w(2D - 0.5)) \sum_i L_i L_i^T}{w + 4wD} \\ &= \Sigma. \end{aligned} \quad (4.35)$$

4.5.3 Linearisation using sigma points

If a multivariate Gaussian \mathbf{X} undergoes a nonlinear transformation $\mathbf{Y} = f(\mathbf{X})$, the resulting distribution \mathbf{Y} is generally not a multivariate Gaussian. With sigma points we can estimate a multivariate Gaussian $\hat{\mathbf{Y}}$ from \mathbf{Y} as follows:

1. parameterise the Gaussian variable \mathbf{X} into sigma points \mathbf{s}_i ,
2. apply $f(\mathbf{s}_i)$ to find the transformed sigma points \mathbf{t}_i , and
3. finally estimate $\hat{\mathbf{Y}}$ by calculating the sample mean and sample covariance of \mathbf{t}_i .

Figure 4.12 shows an example of this linearisation method.

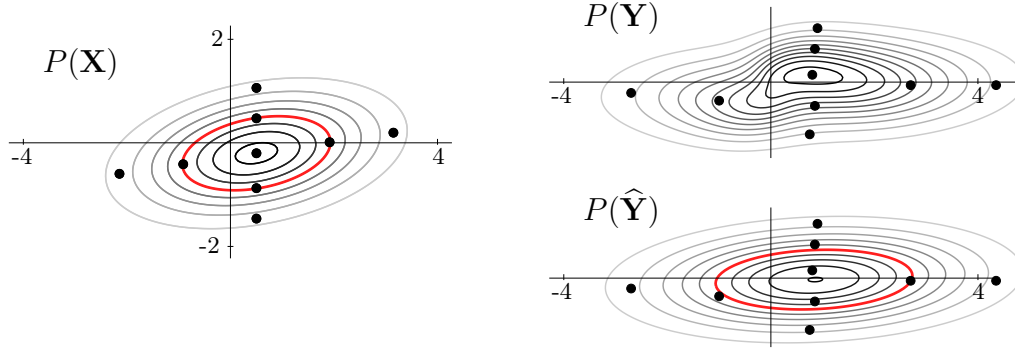


Figure 4.12: An example of the linearisation of a nonlinear transformation. On the left we have the multivariate Gaussian \mathbf{X} and extended sigma points \mathbf{s}_i . On the right we have \mathbf{Y} and \mathbf{t}_i as nonlinear transformations of \mathbf{X} and \mathbf{s}_i respectively. Finally we have the multivariate Gaussian $\hat{\mathbf{Y}}$ as a linear estimation of \mathbf{Y} calculated from the sample mean and covariance of \mathbf{t}_i .

Furthermore, when confronted with a nonlinear transformation $\mathbf{Y} = f(\mathbf{X})$ we are also able to estimate a joint Gaussian $P(\mathbf{X}, \mathbf{Y})$. The idea is to approach the transformation as a black box with the assumption of a linear relationship between the input and output parameters. We parameterise \mathbf{X} into sigma points \mathbf{s}_i and find \mathbf{t}_i as the transformation thereof. By allowing \mathbf{t}_i to represent $\hat{\mathbf{Y}}$ as an estimation of \mathbf{Y} , a joint Gaussian distribution $P(\mathbf{X}, \hat{\mathbf{Y}})$ is found by calculating the sample mean and covariance of both \mathbf{s}_i and \mathbf{t}_i .

Figure 4.13 shows an example of finding such a joint Gaussian. The transformation $\hat{\mathbf{Y}} = \hat{f}(\mathbf{X})$ is incorporated in the joint Gaussian $P(\mathbf{X}, \hat{\mathbf{Y}})$ as follows: we can observe \mathbf{X} to obtain an estimated \mathbf{Y} and vice versa. In essence, we capture a function as a probability distribution: if some of the variables of the distribution are observed, we gain knowledge about the other variables.

The sigma points capture the true mean and covariance of a multivariate Gaussian variable and, when propagated through a nonlinear system, capture the posterior mean and covariance accurately to the third-order Taylor expansion for any nonlinearity [47]. If we compare the unscented Kalman filter, which uses sigma points, to the extended Kalman filter, which uses a first-order Taylor expansion, the computational complexity of the two approaches are of the same order [47]. In Figure 4.14 we show the relationship between the two approaches and how sigma points can provide a more accurate linearisation.

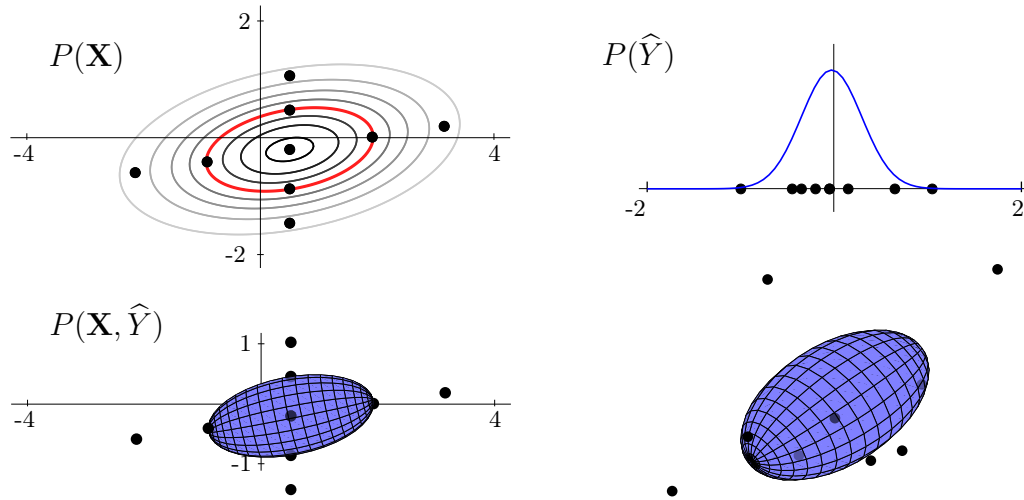


Figure 4.13: An example of estimating a joint Gaussian distribution from a nonlinear transformation. On the top left we have the multivariate Gaussian \mathbf{X} and associated sigma points \mathbf{s}_i . On the top right we have the univariate Gaussian \hat{Y} calculated from the sigma points \mathbf{t}_i , which is a nonlinear transformation of \mathbf{s}_i . On the bottom we have the multivariate Gaussian $P(\mathbf{X}, \hat{Y})$ as a linear estimation of $P(\mathbf{X}, Y)$, calculated from the sample mean and covariance of both \mathbf{s}_i and \mathbf{t}_i . We plot the one standard deviation contour of this 3D distribution as an ellipsoid.

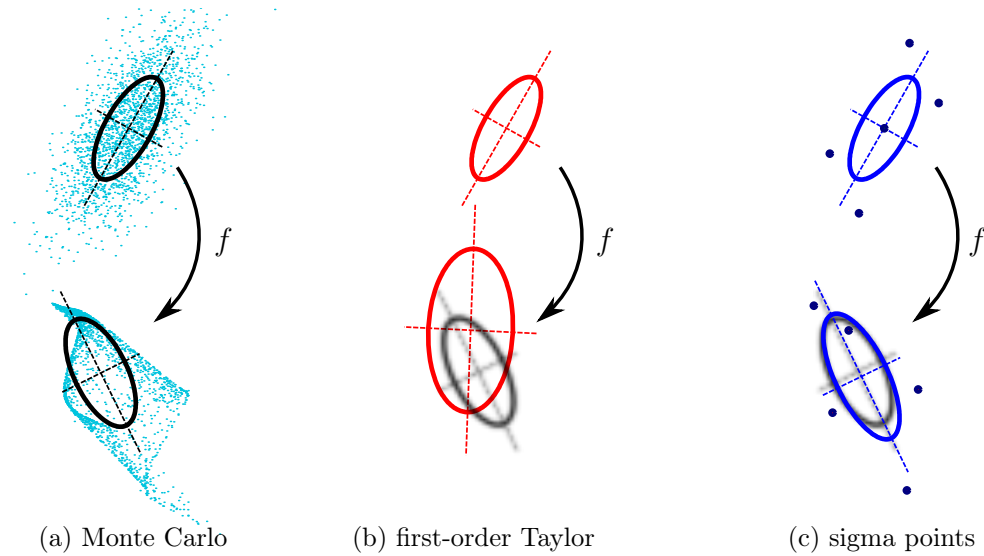


Figure 4.14: A comparison of linearising a function using (a) 2000 Monte Carlo generated samples, (b) a first-order Taylor expansion, and (c) a sigma point formulation. The true Gaussian mean and covariance are taken as the Monte Carlo estimation. It is clear that the sigma point formulation provides a posterior model much closer to the true posterior Gaussian mean and covariance. This figure is redrawn from [47].

Chapter 5

Probabilistic graphical models

In the previous chapter we introduced some general probability theory with a focus on implementing the theory in code. If we have a system with many random variables, the dimensionality of the joint distribution over those variables can become too large to implement. In this chapter we introduce tools to represent and determine the joint distribution with lower-dimensional clusters.

We introduce Bayes networks as a representation of the dependencies between random variables and show how we can use these dependencies to compartmentalise the random variables of a system into clusters. We show how these clusters can be initialised with prior knowledge of the system and be connected in the form of a cluster graph. Finally we introduce belief propagation as a method to pass information between the clusters, for finding a posterior distribution.

Here we focus on a general implementation of this probabilistic approach and return to our structure and motion problem in Chapter 6.

5.1 Hamming code example

For ease of understanding, we explain cluster graphs and belief propagation with the help of an example: the Hamming (7,4) code. The Hamming (7,4) code is a 7-bit encoding of a 4-bit message by adding three parity bits to allow for error correction. These parity bits introduce redundancy to the message so that if a Hamming encoded message is passed through a noisy channel, it is possible to determine with a certain confidence the original message bits.

The encoding works as follows. The four message bits b_1, b_2, b_3, b_4 are the bits that the sender wants to sent to the receiver. The three parity bits b_5, b_6, b_7 are added to the message to ensure that the receiver can validate and possibly correct the received four bits. They are defined to be

$$\begin{aligned} b_5 &= b_1 \oplus b_2 \oplus b_3, \\ b_6 &= b_2 \oplus b_3 \oplus b_4, \\ b_7 &= b_1 \oplus b_3 \oplus b_4, \end{aligned} \tag{5.1}$$

where \oplus is the XOR operator.

When such a message is sent over a noisy channel, the value of each received bit r_1, \dots, r_7 can be analysed to find the original bits b_1, \dots, b_7 with a confidence determined by the system noise and amount of contradictory information.

5.2 Bayes networks

A Bayes network is a type of graphical model that represents the conditional independence between a set of random variables. We use Bayes networks to determine how the random variables can be organised into clusters which can be used for belief propagation, as we will discuss later.

Here is a summary of the probability concepts introduced in Chapter 4:

- Product rule: $P(X_1, X_2) = P(X_1|X_2)P(X_2)$
- Marginalisation: $\sum_{X_2} P(X_1, X_2) = P(X_1)$
- Chain rule: $P(X_1, \dots, X_n) = \prod_{i \in \{1, \dots, n\}} P(X_i | X_{i+1}, \dots, X_n)$
- Independence: $P(X_1, X_2) = P(X_1)P(X_2)$ if $X_1 \perp\!\!\!\perp X_2$
- Conditional independence: $P(X_1, X_2|X_3) = P(X_1|X_3)P(X_2|X_3)$ if $X_1 \perp\!\!\!\perp X_2|X_3$

With B_1, \dots, B_7 and R_1, \dots, R_7 as random variables representing the sent bits and received bits respectively, we may use the chain rule and conditional independence to find the joint probability distribution of the Hamming code (7,4) problem as

$$P(B_1, \dots, B_7, R_1, \dots, R_7) = \quad (5.2)$$

$$P(B_5|B_1, B_2, B_3) P(B_6|B_2, B_3, B_4) P(B_7|B_1, B_3, B_4) \prod_{i=1}^7 P(B_i|R_i) P(R_i).$$

The Bayes network shown in Figure 5.1 is constructed according to the following. A single-line circle around a variable indicates that the variable is unobserved and a double-line circle around a random variable indicates that the variable is observed. The arrows are drawn according to the factors in Equation 5.2, where arrows are drawn from the variables on the right side of a factor's conditional towards the variables on the left side of the factor's conditional.

Note that in most cases a Bayes network is used as a visual guide to map the known relationships between random variables. We therefore use a Bayes network as a tool to determine the factors in the chain rule expansion for a specific set of variables rather than vice versa.

The problem is to estimate the most likely values of B_1, \dots, B_4 given the values of the received bits R_1, \dots, R_7 . That is, we want the states of B_1, \dots, B_4 that

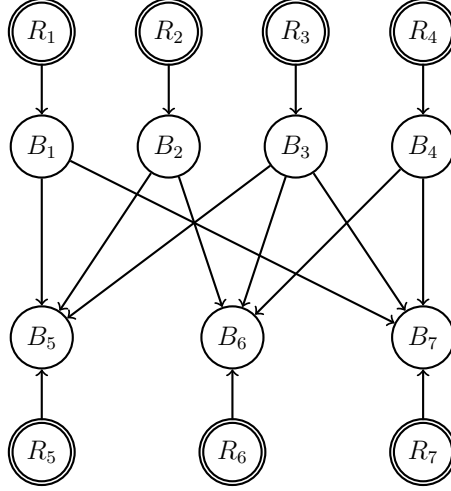


Figure 5.1: A Bayes network describing the Hamming (7,4) code problem. The random variables B_1, \dots, B_4 represent a 4-bit message, B_5, B_6, B_7 represent the parity bits, and R_1, \dots, R_7 represent the received bits. The double-line circles around R_1, \dots, R_7 show that these variables are observed, in this case as $R_1 = r_1, R_2 = r_2, \dots, R_7 = r_7$.

would maximise the conditional distribution

$$P(B_1, \dots, B_4 | R_1 = r_1, \dots, R_7 = r_7). \quad (5.3)$$

If we can estimate the full joint distribution, we can find the conditional in the above expression, and pick the combined state for B_1, \dots, B_4 where it reaches a maximum.

5.3 Cluster graphs

Before we can apply belief propagation and estimate the joint distribution of our system, we first find a cluster graph representing the system. An obvious choice of clusters for a given system is the factors in the chain rule after they are reduced with conditional independences. For the Hamming (7,4) example we can use Equation 5.4 to specify factors for the following 17 subsets of our random variables:

$$\begin{aligned} &\{B_5, B_1, B_2, B_3\}, \{B_6, B_2, B_3, B_4\}, \{B_7, B_1, B_3, B_4\}, \\ &\{B_1, R_1\}, \{B_2, R_2\}, \dots, \{B_7, R_7\}, \\ &\{R_1\}, \{R_2\}, \dots, \{R_7\}. \end{aligned} \quad (5.4)$$

We may use these factors, excluding those that are subsets of others, to construct a cluster graph for the Hamming (7,4) code as shown in Figure 5.2.

A cluster graph \mathcal{T} for a set of factors ϕ_i (such as the factors in Equation 5.4) is an undirected graph with clusters \mathbf{C}_i as nodes, where

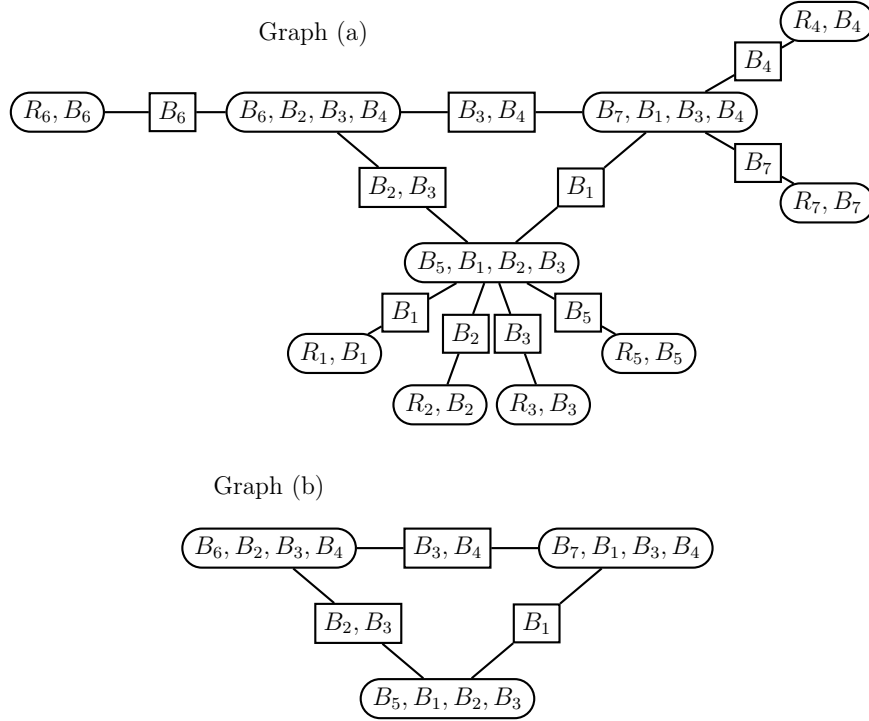


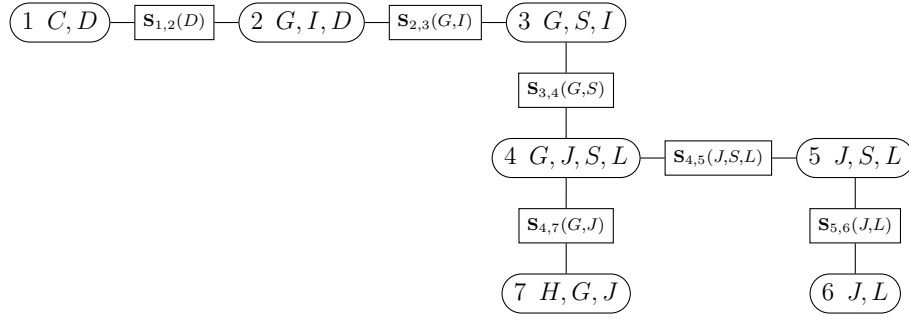
Figure 5.2: A cluster graph of the Hamming (7,4) code problem. Graph (a) is the full cluster graph. Graph (b) shows the clusters that will effectively be involved in loopy belief propagation: if R_i is observed, the factor $\phi_{i+3}(R_i, B_i)$ is effectively reduced to B_i and can be absorbed by the connecting supersets.

1. the clusters are related to the factors such that $\text{Scope}(\phi_i) \subseteq \mathbf{C}_i$,
2. no cluster is a subset of another cluster, that is $\mathbf{C}_i \not\subseteq \mathbf{C}_j$ for all $i \neq j$,
3. the clusters are connected by nonempty sepsets as $\mathbf{S}_{i,j} \subseteq \mathbf{C}_i \cap \mathbf{C}_j$, and
4. the connections abide by the so-called running intersection property [44].

The running intersection property is held by a cluster graph \mathcal{T} if, for any given variable X in that graph, any two clusters \mathbf{C}_i and \mathbf{C}_j containing X have a unique path of sepsets containing X between them (i.e. the sepset connections that contain that particular random variable will form a tree structure). Two examples are given in Figure 5.3.

One method for building such a cluster graph is Du Preez's algorithm [46]. Consider a set of clusters \mathcal{V} and a set of random variables \mathcal{X} as the union of all these clusters. If we look at each random variable $X \in \mathcal{X}$, the sepsets containing that random variable $\mathbf{S}_{i,j}(X, \dots)$ must be formulated in such a way that they are connected as a tree. Du Preez's algorithm formulates these connections by iterating through the random variables $X \in \mathcal{X}$ and connecting each cluster $\mathbf{C} \in \mathcal{V}$ containing X in a tree structure \mathcal{T}_X . Finally all these tree structures \mathcal{T}_X are superimposed to find the cluster graph \mathcal{T} .

Graph (a)



Graph (b)

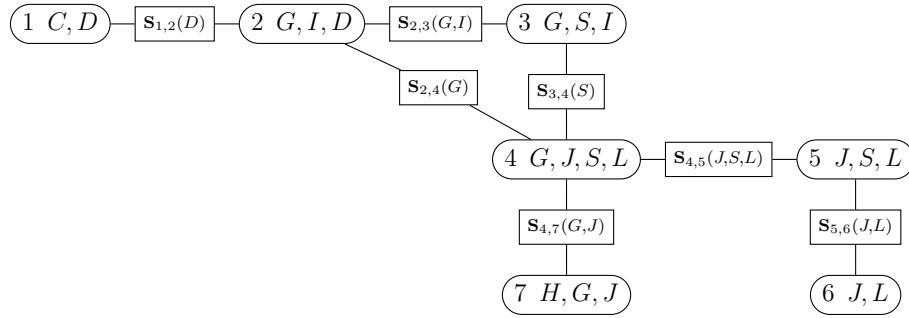


Figure 5.3: Examples of the running intersection property. Graph (a) and (b) are constructed from the same clusters, but are connected with different sepsets, yet both these graphs abide by the running intersection property.

A detailed outline of Du Preez's algorithm is presented in Algorithm 2 and a visual illustration to the process is shown in Figure 5.4. In this figure we have B as the current random variable for which we build the tree and therefore the illustrated clusters are the subset of clusters from \mathcal{V} containing B .

An alternative to using cluster graphs is factor graphs. A factor graph is constructed by

- assigning some clusters to the main nodes of the graph (called factors),
- creating univariate nodes, one for each variable in the system, and
- connecting all univariate nodes to all factors with that node in its scope.

In the current literature it seems as if factor graphs are more widely used than cluster graphs. This might be due to the fact that factor graphs are trivially constructed while cluster graphs require careful planning and some heuristics to be constructed properly. A comparative study by Du Plessis [48] found cluster graphs to converge quicker than factor graphs with seemingly the same accuracy. This can largely be attributed to the fact that multivariate sepsets, found in cluster graphs, allow for information about interactions between variables to be propagated which effectively reduces the number of messages that

needs to be sent before convergence. We therefore keep our focus on cluster graphs.

Algorithm 2: Du Preez's algorithm

Input: Set of clusters $\mathcal{V} = \{C_1, \dots, C_n\}$ where $C_i \not\subseteq C_j$ for all $i \neq j$

Output: Cluster graph \mathcal{T}

```

1:  $\mathcal{S} := \{\}$ 
2: for  $X \in \bigcup\{C_i \in \mathcal{V}\}$  do
3:    $\mathcal{V}_X :=$  clusters in  $\mathcal{V}$  containing  $X$ 
    $\triangleright$  Fully connect  $\mathcal{V}_X$  with weights
4:    $\mathcal{W}_X := \{w_{i,j} = |C_i \cap C_j| : C_i, C_j \in \mathcal{V}_X, i \neq j\}$ 
    $\triangleright$  Add tiebreakers to weights
5:   for  $C_i \in \mathcal{V}_X$  do
6:      $t_i := \left\{ w_{i,j} : w_{i,j} \equiv \max(\mathcal{W}_X) \wedge w_{i,j} \in \right.$ 
        $\left. \text{all weights in } \mathcal{W}_X \text{ adjacent to } C_i \right\}$ 
7:   end for
8:   for  $C_i$  in  $\mathcal{V}_X$  do
9:     add  $t_i$  to each  $w_{i,j}$  adjacent to  $C_i$ 
10:  end for
    $\triangleright$  Extend  $\mathcal{S}$ 
11:  for each  $(i, j)$  link in the maximum spanning tree of  $\{\mathcal{V}_X, \mathcal{W}_X\}$  do
12:    if  $S_{i,j}$  exists as  $\in \mathcal{S}$  then
13:       $S_{i,j} := S_{i,j} \cup \{X\}$ 
14:    else
15:      add  $S_{i,j} = \{X\}$  to  $\mathcal{S}$ 
16:    end if
17:  end for
18:   $\mathcal{T} := \mathcal{V}$  connected as a cluster graph with sepsets  $\mathcal{S}$ 
19: end for

```

Input Although we state that $C_i \not\subseteq C_j$ where $i \neq j$, the algorithm can be adjusted to allow the case where \mathcal{V} has subsets: in line 4 where we compare clusters to find their intersections, we can remove any cluster that is a subset of another cluster by allowing the larger cluster to absorb the smaller cluster.

Line 11 We find a maximum spanning tree using Prim's algorithm [49]. This is a greedy algorithm for constructing a spanning tree from a connected weighted graph. The algorithm can be used with a sophisticated Fibonacci heap to obtain a remarkable time complexity of $\mathcal{O}(|E| + |V| \log |V|)$ [50] and is available in the C++ boost library.

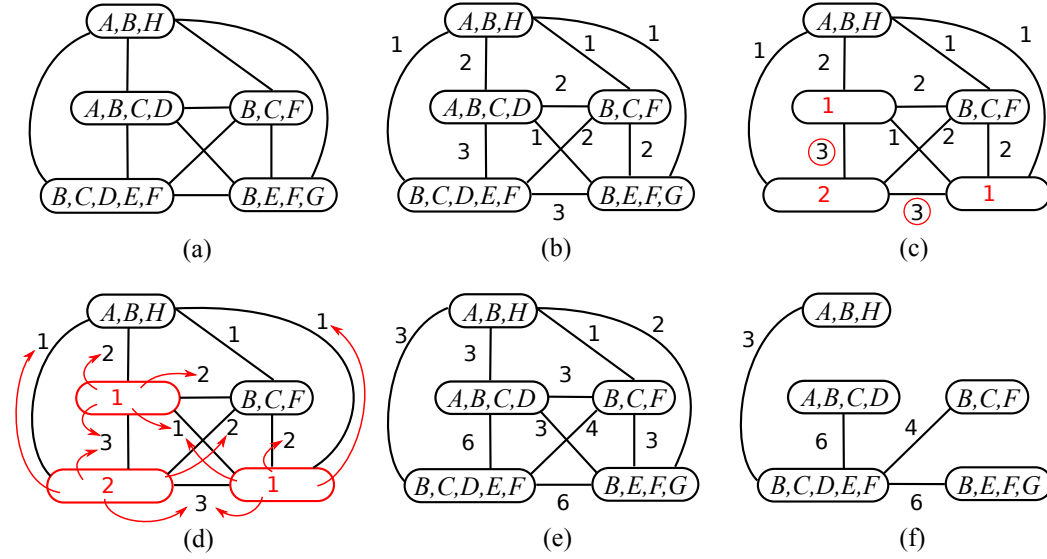


Figure 5.4: An illustration of the behaviour inside the main loop of Du Preez's algorithm, as an example of constructing an intermediate spanning tree. Steps (a) to (f) are as follows: (a) we have a fully connected graph for all clusters containing B , (b) we choose the edge weights according to cluster intersections, (c) we show the cluster tiebreakers t_i in red (this is a count of all the weights adjacent to the cluster with a weight equal to the maximum weight) and encircle the maximum weights also in red, (d) we add the cluster tiebreakers to all the weights to obtain (e), (f) we use Prim's algorithm [49] to find a maximum spanning tree.

5.4 Message passing

We now show how to pass information between the nodes of a cluster graph in order to reach a consensus about the random variables involved in the graph. For example, if we have a cluster which represents a conditional space, the information about the underlying distributions of the conditioned variables are unknown. With message passing this information can be propagated from other clusters and be integrated in the cluster's knowledge about the variables.

Before we can apply loopy belief propagation and obtain the joint probability distribution of the Hamming (7,4) example, we must walk through the tools necessary to implement this algorithm. Belief propagation yields exact results on a tree structured cluster graph, but for graphs containing loops it is tailored as loopy belief propagation, a heuristic that iteratively estimates the posterior.

5.5 Belief propagation

We first introduce belief propagation on a tree structured cluster graph (such as in Figure 5.5) and then return to the Hamming (7,4) code example to show how it can be applied on a loopy graph.

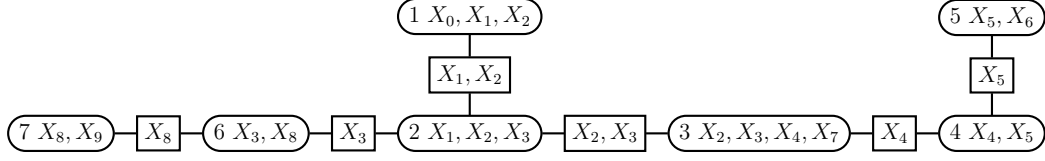


Figure 5.5: A tree structured cluster graph. This graph is used as an example for exact inference using belief propagation.

We use the following two concepts as part of belief propagation: $\beta_i(\mathbf{C}_i)$ is the cluster belief and $\delta_{i \rightarrow j}(\mathbf{S}_{i,j})$ is the sepset belief. We often drop the brackets and refer to the beliefs directly as β_i and $\delta_{i \rightarrow j}$. The beliefs are formulated as follows [51]:

$$\beta_i = \psi_i \prod_{k \in \text{Adj}(i)} \delta_{k \rightarrow i} \quad \text{and} \quad (5.5)$$

$$\delta_{i \rightarrow j} = \sum_{\mathbf{C}_i \setminus \mathbf{S}_{i,j}} \psi_i \prod_{k \in (\text{Adj}(i) \setminus \{j\})} \delta_{k \rightarrow i}, \quad (5.6)$$

with \setminus the set difference, $\text{Adj}(i)$ the set of indices for the clusters adjacent to cluster i , and $\psi_i(\mathbf{C}_i)$ the initial factor assigned to \mathbf{C}_i (the prior).

The belief of a cluster represents the posterior distribution over the random variables in that cluster. For example, if a cluster $\mathbf{C}_1 = \{X_0, X_1, X_2\}$ is initialised with a factor ψ_1 representing $P(X_0 | X_1 = x_1, X_2 = x_2)$, a conditional distribution, the belief of that cluster β_1 will then represent $P(X_0, X_1, X_2)$, the joint distribution over $\{X_0, X_1, X_2\}$. Note that if we use factor tables, as discussed in Section 4.2, in our calculations, the beliefs would need to be normalised in order to be probability distributions.

The procedure for finding all the beliefs of a tree structured cluster graph returns exact beliefs. The steps are outlined in Algorithm 3 [44], which is known as sum-product message passing. To give a clear example, if we apply Algorithm 3 on the cluster graph in Figure 5.5 the following passing order is valid:

1. we first start with the end node (5),

$$\delta_{5 \rightarrow 4} := \sum_{X_6} \psi_5$$

$$\delta_{4 \rightarrow 3} := \sum_{X_5} \psi_4 \delta_{5 \rightarrow 4}$$

$$\delta_{3 \rightarrow 2} := \sum_{X_4, X_7} \psi_3 \delta_{4 \rightarrow 3}$$

2. then we go to the end node (7),

$$\delta_{7 \rightarrow 6} := \sum_{X_9} \psi_7$$

$$\delta_{6 \rightarrow 2} := \sum_{X_8} \psi_6 \delta_{7 \rightarrow 6}$$

3. then the end node (1),

$$\delta_{1 \rightarrow 2} := \sum_{X_0} \psi_1$$

4. and finally, we reverse the above order to spread the consensus back across all the nodes,

$$\delta_{2 \rightarrow 1} := \sum_{X_3} \psi_2 \delta_{3 \rightarrow 2} \delta_{6 \rightarrow 2}$$

$$\delta_{2 \rightarrow 6} := \sum_{X_1, X_2} \psi_2 \delta_{1 \rightarrow 2} \delta_{3 \rightarrow 2}$$

$$\delta_{6 \rightarrow 7} := \sum_{X_3} \psi_6 \delta_{2 \rightarrow 6}$$

$$\delta_{2 \rightarrow 3} := \sum_{X_4, X_7} \psi_2 \delta_{1 \rightarrow 2} \delta_{6 \rightarrow 2}$$

$$\delta_{3 \rightarrow 4} := \sum_{X_5} \psi_3 \delta_{2 \rightarrow 3}$$

$$\delta_{4 \rightarrow 5} := \sum_{X_6} \psi_4 \delta_{3 \rightarrow 4}.$$

After all the messages are calculated, we find the belief of each of the clusters as

$$\beta_i := \psi_i \prod_{j \in \text{Adj}(i)} \delta_{j \rightarrow i}. \quad (5.7)$$

If, for example, we want the beliefs for cluster C_2 , we obtain

$$\beta_2 := \psi_2 \delta_{1 \rightarrow 2} \delta_{6 \rightarrow 2} \delta_{3 \rightarrow 2}. \quad (5.8)$$

Algorithm 3: Cluster tree sum-product message passing

Input: Cluster tree \mathcal{T} with edges \mathcal{E} and sepsets \mathcal{S}

Output: All the beliefs β_i corresponding to the clusters C_i

- ▷ Find the sepset beliefs
- 1: **while** there is a C_i ready to transmit to C_j **do**
- 2: $\delta_{i \rightarrow j} := \sum_{C_i \setminus \mathbf{s}_{i,j}} \psi_i \prod_{k \in (\text{Adj}(i) \setminus \{j\})} \delta_{k \rightarrow i}$
- 3: **end while**
- ▷ Find the cluster beliefs
- 4: **for** each cluster C_i **do**
- 5: $\beta_i := \psi_i \prod_{k \in \text{Adj}(i)} \delta_{k \rightarrow i}$
- 6: **end for**

Line 1 A cluster C_i is ready to transmit a message to C_j if C_i has received messages from all of the neighbouring clusters excluding C_j [44].

5.6 Loopy belief propagation

In the case where a cluster graph contains loops, all the messages in the loop rely on other messages in that loop. Our tree structure message passing scheme can therefore be caught in a deadlock, where no cluster is ready to transmit

while not all clusters have received a message yet. We can heuristically solve this problem by passing messages that are not ready to transmit continually until the sepset messages converge. Although it is not proven that such a message passing scheme will converge or that the joint probability obtained will be a good representation of the real joint distribution, if implemented effectively such a scheme typically returns practical results [44].

One of the problems of such a loopy belief propagation scheme is to choose a suitable message schedule. Koller [44] has found synchronous message passing (where all messages are updated at once) to be far from optimal for most cases. This is both in terms of converging to an optimal solution, and in the number of messages required for this convergence. Asynchronous message passing on the other hand is more efficient in converging. A proper message passing schedule is the key to improving convergence time and to improve the chances of converging to the optimal solution.

Our message passing schedule prioritises information spread from messages that underwent large updates. The procedure is as follows: after a message $\delta_{i \rightarrow j}$ is propagated, we add all the messages $\delta_{j \rightarrow k}$ where $k \in \text{Adj}(j) \setminus \{i\}$ to a priority queue with the weight as some update error metric. By doing so, we prioritise the parts of the graph that are likely far from stable convergence. The sum-product message passing scheme adjusted for a loopy graph is shown in Algorithm 4.

When working with discrete probability distributions, we use for our update metric the Kullback-Leibler divergence to estimate the error between a message and the same message from a previous step. The Kullback-Leibler divergence $D_{KL}(P||Q)$ is a measurement of information lost when P is approximated by Q :

$$D_{KL}(P||Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}. \quad (5.9)$$

One should note that the behaviour of the Kullback-Leibler divergence is non-symmetric, i.e. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$, and is therefore not a true distance or error metric. Since the Kullback-Leibler divergence does contain information about the similarity between two probability distributions, the naive approach of choosing this divergence as an error metric still works well for this application [52].

When working with Gaussian distributions, we take for our update metric the average of the Mahalanobis distance [53] between the one distribution and the mean of the other distribution and vice versa. Alternative metrics also exist such as the Kullback-Leibler divergence for Gaussian distributions [44].

Algorithm 4: Cluster graph sum-product message passing**Input:** Cluster graph \mathcal{T} with edges \mathcal{E} and sepsets \mathcal{S} **Output:** All the beliefs β_i corresponding to the clusters \mathbf{C}_i

```

    ▷ Initialise priority queue with a minimally connected  $(i, j)$  of priority  $\infty$ 
1:  $q := \{\infty : (i, j), \infty : (j, i)\}$ , where  $(i, j)$  minimises  $|\text{Adj}(i) + \text{Adj}(j)|$ 
    ▷ Initialise the cluster messages
2: for each edge  $\mathbf{E}_{i,j}$  in  $\mathcal{E}$  do
3:    $\delta_{i \rightarrow j} := \mathbf{1}$ 
4:    $\delta_{j \rightarrow i} := \mathbf{1}$ 
5: end for
    ▷ Message passing
6: while  $q$  is not empty do
7:    $(i, j) := q.\text{pop}()$ 
8:    $\delta_{\text{prev}} := \delta_{i \rightarrow j}$ 
9:    $\delta_{i \rightarrow j} := \text{norm}\left(\sum_{\mathbf{C}_i \setminus \mathbf{s}_{i,j}} \psi_i \prod_{k \in (\text{Adj}(i) - \{j\})} \delta_{k \rightarrow i}\right)$  ▷ Update message
10:  priority := error( $\delta_{\text{prev}}, \delta_{i \rightarrow j}$ )
11:  for  $k \in \text{Adj}(j) - \{i\}$  do
12:    if  $(j, k) \in q$  then
13:      remove  $(j, k)$  from  $q$ 
14:    end if
15:    if priority > chosen threshold then
16:       $q.\text{push}(\text{priority}: (j, k))$ 
17:    end if
18:  end for
19: end while
    ▷ Find the posterior beliefs
20: for each cluster  $\mathbf{C}_i$  do
21:    $\beta_i := \psi_i \prod_{k \in \text{Adj}(i)} \delta_{k \rightarrow i}$ 
22: end for

```

Line 1–5 An alternative is to initialise the priority queue by setting $\delta_{i \rightarrow j} := \text{norm}\left(\sum_{\mathbf{C}_i \setminus \mathbf{s}_{i,j}} \psi_i\right)$ for all (i, j) edges. Experimentally we have found this to lead to faster convergence [46].

Line 9 To apply damping to the messages when using probability tables, we add the following line after line 9:

$$\delta_{i \rightarrow j} := (1 - \lambda)(\delta_{\text{prev}}) + (\lambda)(\delta_{i \rightarrow j}),$$

where λ is the damping factor.

Line 15 We assume that temporary local convergence has been reached when a message update is below a chosen threshold. If all the messages in the system is below this threshold, convergence is reached.

5.7 Example of solving a Hamming code

Initialising the system: We initialise the cluster graph in Figure 5.2 by incorporating the logic from Equation 5.1 into the associated factors in Equation 5.4. A factor such as $\psi_1(B_5, B_1, B_2, B_3)$, associated with the conditional distribution $P(B_5|B_1, B_2, B_3)$, is a 4-dimensional discrete probability factor that can be indexed as $\psi_1[i, j, k, l]$, with the relationship between ψ_1 and $P(B_5|B_1, B_2, B_3)$ as such: the value returned by $\psi_1[0, 1, 1, 0]$ is the probability that

$$P(B_5=0|B_1=1, B_2=1, B_3=0),$$

when ψ_1 is normalised appropriately.

We start by considering the logic inherent in $b_5 = b_1 \oplus b_2 \oplus b_3$ to initialise ψ_1 using the following:

```

1: for  $i \in \{0, 1\}$  do
2:   for  $j \in \{0, 1\}$  do
3:     for  $k \in \{0, 1\}$  do
4:       for  $l \in \{0, 1\}$  do
5:         if  $i = j \oplus k \oplus l$  then
6:            $\psi_1[i, j, k, l] := 100\%$ 
7:         else
8:            $\psi_1[i, j, k, l] := 0\%$ 
9:         end if
10: end for, for, for, for
```

We simply assign $\psi_2 := \text{copy}(\psi_1)$ and $\psi_3 := \text{copy}(\psi_1)$.

The factors $\psi_4(B_1, R_1), \dots, \psi_{10}(B_7, R_7)$ are associated with the conditional distributions $P(B_1|R_1), \dots, P(B_7|R_7)$. We might choose somewhat arbitrarily that the probability of a received bit being equal to a transmitted bit is 90%, so that

```

1:  $\psi_4[0, 0] := \psi_4[1, 1] := 90\%$ 
2:  $\psi_4[0, 1] := \psi_4[1, 0] := 10\%$ 
```

We similarly assign $\psi_i := \text{copy}(\psi_4)$ for $i = 5, \dots, 10$.

Now that we have a cluster graph with all the factors initialised, we can observe some of the random variables of the system and run loopy belief propagation (Algorithm 4) to find the distribution over the other random variables.

Observe and solve: Suppose the message “1010” (with the Hamming (7,4) message being “1010010”) is sent over a noisy channel and the received bit string is observed as “1110010”. By setting the observations in our system as

$$R_1=1, R_2=1, R_3=1, R_4=0, R_5=0, R_6=1, R_7=0,$$

the factors ψ_4, \dots, ψ_{10} are reduced to $\psi_4(B_1), \dots, \psi_{10}(B_7)$, with values such as $\psi_4[1]=90\%$, $\psi_5[1]=90\%$, \dots , $\psi_9[1]=90\%$, $\psi_{10}[1]=10\%$. By running belief propagation on the system, we obtain a posterior distribution over the system

from which we can pick the combined state for B_1, \dots, B_7 where it reaches a maximum.

β_4 , the posterior of factor ψ_4 after running belief propagation on the clusters, is

B_5	B_1	B_2	B_3	value
0	0	0	0	0.00000101
0	0	0	1	0.00000000
0	0	1	0	0.00000000
0	0	1	1	0.00991713
0	1	0	0	0.00000000
0	1	0	1	0.9800115
0	1	1	0	0.00009991
0	1	1	1	0.00000000
1	0	0	0	0.00000000
1	0	0	1	0.00000001
1	0	1	0	0.00010000
1	0	1	1	0.00000000
1	1	0	0	0.00000101
1	1	0	1	0.00000000
1	1	1	0	0.00000000
1	1	1	1	0.00986940

Note that β_4 is normalised. Furthermore, by querying the marginals of all the random variables in the system, we obtain

	B_1	B_2	B_3	B_4	B_5	B_6	B_7
“0”	01.01%	98.01%	00.03%	98.99%	99.00%	00.99%	99.00%
“1”	98.99%	01.99%	99.97%	01.01%	01.00%	99.01%	01.00%

With this approach we therefore manage to successfully retrieve the original message “1010010” with a very high certainty even though one of the received bits is corrupted.

Chapter 6

Our probabilistic formulation

Our goal for this chapter is to combine the theory presented on multiview geometry (Chapter 3) and the theory presented on probabilistic techniques (Chapters 4 and 5) towards solving the structure and motion problem. The general idea is to use prior information about the camera poses and scene structure and integrate this information with observations (in the form of feature correspondences) to obtain posterior knowledge about the system. We do so by

- specifying the parameters of the system,
- specifying the relationships between these parameters as well as prior distributions for these parameters,
- categorising the parameters into clusters,
- constructing a cluster graph from these clusters,
- and finally running belief propagation on the cluster graph to obtain a posterior distribution.

These steps result in a posterior distribution over all the parameters of the system which, in our case, amounts to a solution of a given structure and motion problem. Our approach is not only specific to this problem, but can be viewed as a general approach to solving systems with nonlinear dependencies.

6.1 The parameters of the system

We first need to specify all the parameters and represent them as random variables. We derive these parameters directly from the pinhole camera model as discussed in Section 3.2.

To recapitulate the pinhole camera model, the projection of a 3D point \mathbf{X} to a 2D point \mathbf{x} on the projection plane of a camera can be expressed in homogeneous coordinates as

$$\tilde{\mathbf{x}} = P\tilde{\mathbf{X}}. \quad (6.1)$$

The camera matrix \bar{P} is of the form

$$\bar{P} = R \begin{bmatrix} I & | & -\mathbf{C} \end{bmatrix}, \quad (6.2)$$

where R is the 3D rotation of the camera relative to a fixed world coordinate system and \mathbf{C} is its 3D translation (in Euclidean coordinates). Recall that we consider $\tilde{\mathbf{x}}$ to be normalised image coordinates where the effects of the camera calibration matrix K have been removed.

For the purposes of our system, we will represent a camera matrix \bar{P} as a single vector containing all the necessary camera parameters:

$$\mathbf{p} = \begin{bmatrix} C_x & C_y & C_z & \theta_x & \theta_y & \theta_z \end{bmatrix}^T, \quad (6.3)$$

where $\mathbf{C} = [C_x \ C_y \ C_z]^T$, and θ_x , θ_y and θ_z are three Euler angles constructed from the rotation expressed by R . Also, we will consistently use n and i to index cameras and 3D points respectively. For example, 3D point i is captured by camera n as

$$\tilde{\mathbf{x}}_n^i = \bar{P}_n \tilde{\mathbf{X}}^i. \quad (6.4)$$

We also present the function $\mathbf{x} = f(\mathbf{p}, \mathbf{X})$ as an equivalent to Equation 6.1, but with Euclidean parameters as

function $f(\mathbf{p}, \mathbf{X})$:

- 1: $\bar{P} :=$ build camera matrix from \mathbf{p}
- 2: $\tilde{\mathbf{x}} := \bar{P} \begin{bmatrix} X_1 & X_2 & X_3 & 1 \end{bmatrix}^T$
- 3: $\mathbf{x} := \begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 \end{bmatrix}^T \div \tilde{x}_3$
- 4: **return** \mathbf{x}

Recall that in Chapter 4 we denote random variables using uppercase and normal variables using lowercase. In an attempt to comply with this notation, we henceforth denote the camera projection presented in Equation 6.4 with lowercase vectors:

$$\tilde{\mathbf{x}}_n^i = \bar{p}_n \mathbf{x}^i, \quad (6.5)$$

and denote its equivalent using random variables as

$$\mathbf{Y}_n^i = f(\mathbf{P}_n, \mathbf{X}^i), \quad (6.6)$$

where \mathbf{Y}_n^i , \mathbf{P}_n and \mathbf{X}^i represent a projected point, camera parameters and a 3D point respectively. These random variables can be observed as

$$\mathbf{Y}_n^i = \mathbf{y}_n^i, \ \mathbf{P}_n = \mathbf{p}_n, \ \text{and} \ \mathbf{X}^i = \mathbf{x}^i. \quad (6.7)$$

6.2 The dependencies within the system

Our goal is to determine the joint distribution over all the random variables of the system. We do so by combining all the available information of the system,

such as the parameters, their dependencies and prior knowledge about their states.

A small example of a Bayes network for a structure and motion problem is shown in Figure 6.1. It describes the conditional dependencies between the variables, and within these dependencies are the projective relationships between our parameters. Unlike most linear systems, these relationships are not straightforward to model or to invert.

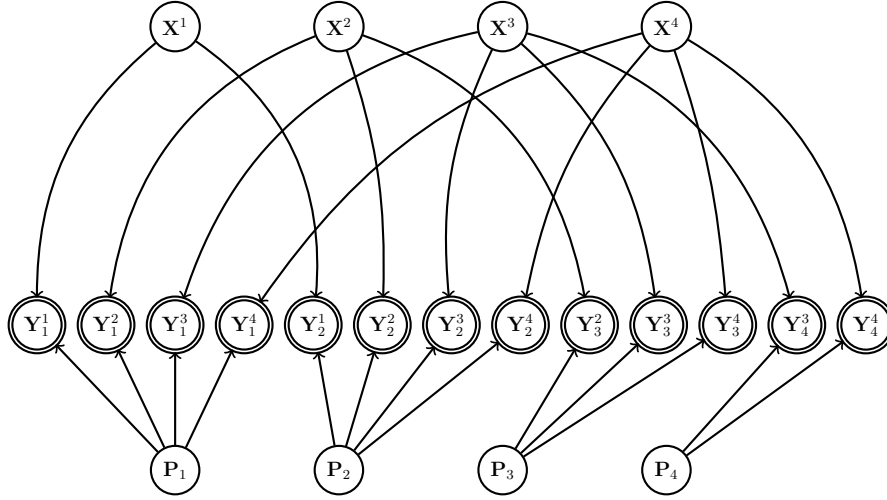


Figure 6.1: A Bayes network describing the relationships between points of a 3D structure \mathbf{X}^i , camera parameters \mathbf{P}_n and the observed projections \mathbf{Y}_n^i . In this example not all the points are visible to all the cameras.

6.3 Nonlinear projective geometry

The relationships between the parameters can be incorporated into the system using random distributions and linear relationships. We will therefore attempt to capture all projective geometry through correlation within multivariate random variables.

For mathematical convenience, and also practical feasibility, we decide to model these relationships by assuming that the observed variables, namely the projected points \mathbf{Y}_n^i , are Gaussian. For every projected point the mean is taken as the coordinates themselves and the standard deviation is taken as the estimated deviation caused by measurement noise (in pixels).

Furthermore, if we have linear relationships between the measurements and the unknown parameters, we will be able to model all the parameters as Gaussian. A linear transformation on a Gaussian returns yet another Gaussian as discussed in Section 4.4.

For our investigation, we begin by representing all the transformations in our system as factors: for every \mathbf{Y}_n^i we find a factor containing all the parameters involved in its projection $\mathbf{Y}_n^i = f(\mathbf{P}_n, \tilde{\mathbf{X}}^i)$. If we try to find this factor as suggested in Section 4.4, some difficulties arise since Equation 6.5 does not follow the specified form for linearity:

$$\mathbf{Y} = A^T \mathbf{X} + \mathbf{c}, \quad (6.8)$$

where \mathbf{Y} and \mathbf{X} are linearly related Gaussians and A and \mathbf{c} are known. The 3D point and the projected point in Equation 6.5 are presented in homogeneous form and the Euclidean equivalent function $\mathbf{Y}_n^i = f(\mathbf{P}_n, \mathbf{X}^i)$ is nonlinear. The camera parameters are also modelled as random variables, which further complicates the system.

To explain how our structure and motion problem is parameterised probabilistically, we first use a lower-dimensional analogy of the pinhole camera model, one that projectively transforms 2D space to 1D space as

$$\mathbf{Y}_n^i = f_{2D}(\mathbf{P}_n, \mathbf{X}^i), \quad (6.9)$$

with \mathbf{X}^i now a 2D point, \mathbf{P}_n the projection parameters of a 2D camera and \mathbf{Y}_n^i a projection of \mathbf{X}^i using the n^{th} camera. We use this 2D analogy for illustration purposes in this chapter, but implement the full 3D case for our experiments in Chapter 7.

To visualise the nature of some of the nonlinearities found in such a system, we show an example of an inverse projection and triangulation in Figure 6.2. In the example we have two cameras \mathbf{p}_1 and \mathbf{p}_2 along with the projections \mathbf{Y}_1 and \mathbf{Y}_2 of the point \mathbf{X} . The positions of the cameras are fixed, \mathbf{Y}_1 and \mathbf{Y}_2 are Gaussian variables taken as measurements and the distribution over \mathbf{X} is found by inverting these projections. On the left hand side of Figure 6.2 we have a fixed camera \mathbf{p}_1 and the univariate Gaussian \mathbf{Y}_1 , along with the 2D non-Gaussian \mathbf{X} found by back-projecting \mathbf{Y}_1 . On the right hand side of the figure, we have another camera \mathbf{p}_2 and projection \mathbf{Y}_2 . By combining the back-projections of \mathbf{Y}_1 and \mathbf{Y}_2 , we find the posterior distribution over \mathbf{X} .

It is clear from this example that although starting with Gaussian distributions, nonlinearities in the system hinder dependent variables to also be modelled as Gaussian. Therefore, our system cannot be modelled as Gaussian by following a direct approach.

6.4 Linearising projective geometry

A solution to the problem of a nonlinear system is to make use of a sigma point formulation as explained in Section 4.5. Doing so allows us to model a linear estimate of the system as Gaussian. In Section 3.4 an algebraic solution to multiview geometry is shown. Here we show how the system can be estimated probabilistically with the aid of sigma points.

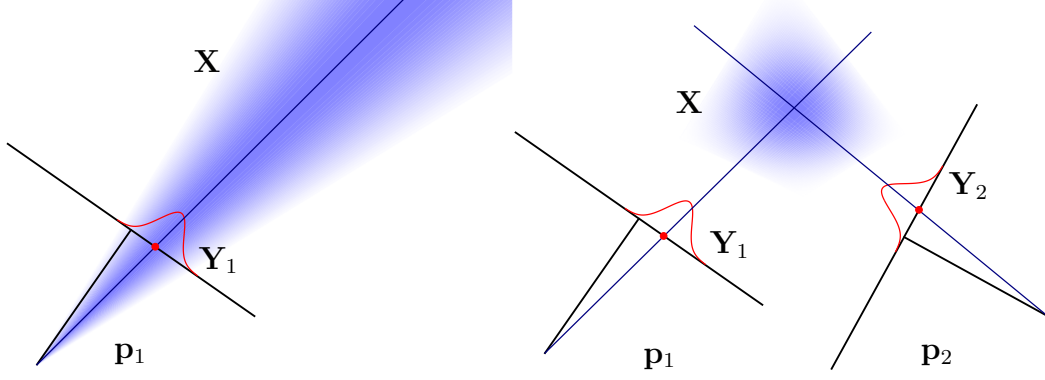


Figure 6.2: An example of nonlinearities arising from projection: On the left we have a static camera \mathbf{p}_1 and a projection \mathbf{Y}_1 modelled as a Gaussian (in red). From this we find the inverse projection \mathbf{X} (in blue). On the right we have cameras \mathbf{p}_1 and \mathbf{p}_2 with the projections \mathbf{Y}_1 and \mathbf{Y}_2 modelled as Gaussians (in red), and the combined inverse projection \mathbf{X} (in blue). It is clear that the inverse projections fail to take the shape of a Gaussian, even when the projections are modelled as such.

6.4.1 Triangulation example

Suppose we are given the poses of two fixed cameras \mathbf{p}_1 and \mathbf{p}_2 , two projections \mathbf{Y}_1 and \mathbf{Y}_2 modelled as Gaussians, and a prior \mathbf{X} for the possible location of the posterior triangulated point represented by $\hat{\mathbf{X}}$.

The distribution over $\hat{\mathbf{X}}$ can be obtained through either

- obtaining $P(\mathbf{X}, \mathbf{Y}_1, \mathbf{Y}_2 | \mathbf{P}_1 = \mathbf{p}_1, \mathbf{P}_2 = \mathbf{p}_2)$ as a prior distribution using sigma points and then observing \mathbf{Y}_1 and \mathbf{Y}_2 in order for the posterior over $\hat{\mathbf{X}}$ to be taken as $P(\mathbf{X} | \mathbf{Y}_1 = \mathbf{y}_1, \mathbf{Y}_2 = \mathbf{y}_2, \mathbf{P}_1 = \mathbf{p}_1, \mathbf{P}_2 = \mathbf{p}_2)$, or
- obtaining the conditionals $P(\mathbf{X}, \mathbf{Y}_1 | \mathbf{P}_1 = \mathbf{p}_1)$ and $P(\mathbf{X}, \mathbf{Y}_2 | \mathbf{P}_2 = \mathbf{p}_2)$ as prior distributions using sigma points, observing \mathbf{Y}_1 and \mathbf{Y}_2 respectively and combining the two results to find the posterior over $\hat{\mathbf{X}}$.

We illustrate the latter of these two approaches. Suppose we have some prior distribution over \mathbf{X} (obtained for example from another system or from guess-work). In Figure 6.3 we parameterise this distribution as sigma points and transform these points using the parameters \mathbf{p}_1 and \mathbf{p}_2 . As a result we now have prior distributions for \mathbf{X} , \mathbf{Y}_1 and \mathbf{Y}_2 , and also $P(\mathbf{X}, \mathbf{Y}_1 | \mathbf{P}_1 = \mathbf{p}_1)$ and $P(\mathbf{X}, \mathbf{Y}_2 | \mathbf{P}_2 = \mathbf{p}_2)$.

In Figure 6.4 we triangulate two features point \mathbf{y}_1^1 and \mathbf{y}_2^1 using these priors. We first convert the measurements to narrow Gaussian distributions, \mathbf{Y}_1^1 and \mathbf{Y}_2^1 , to allow for measurement noise. These Gaussians are then used as observations to yield $P(\mathbf{X} | \mathbf{Y}_1^1, \mathbf{p}_1)$ and $P(\mathbf{X} | \mathbf{Y}_2^1, \mathbf{p}_2)$. Finally we combine these two results to find the posterior triangulated point $\hat{\mathbf{X}}$.

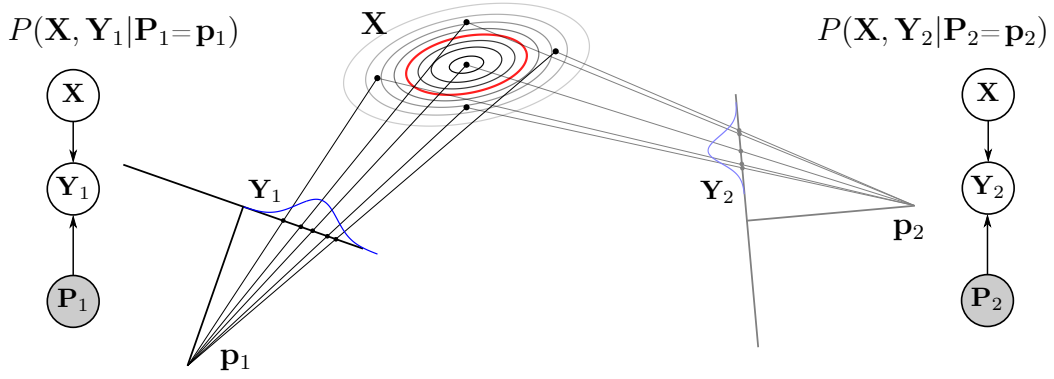


Figure 6.3: The Gaussian parameterisation of points: the Gaussian distribution \mathbf{X} is parameterised as sigma points and transformed using the parameters \mathbf{p}_1 and \mathbf{p}_2 to obtain distributions over \mathbf{Y}_1 and \mathbf{Y}_2 respectively. The joint Gaussians $P(\mathbf{X}, \mathbf{Y}_1 | \mathbf{P}_1 = \mathbf{p}_1)$ and $P(\mathbf{X}, \mathbf{Y}_2 | \mathbf{P}_2 = \mathbf{p}_2)$ can be found by calculating the sample mean and covariance of the original sigma points and their transformations.

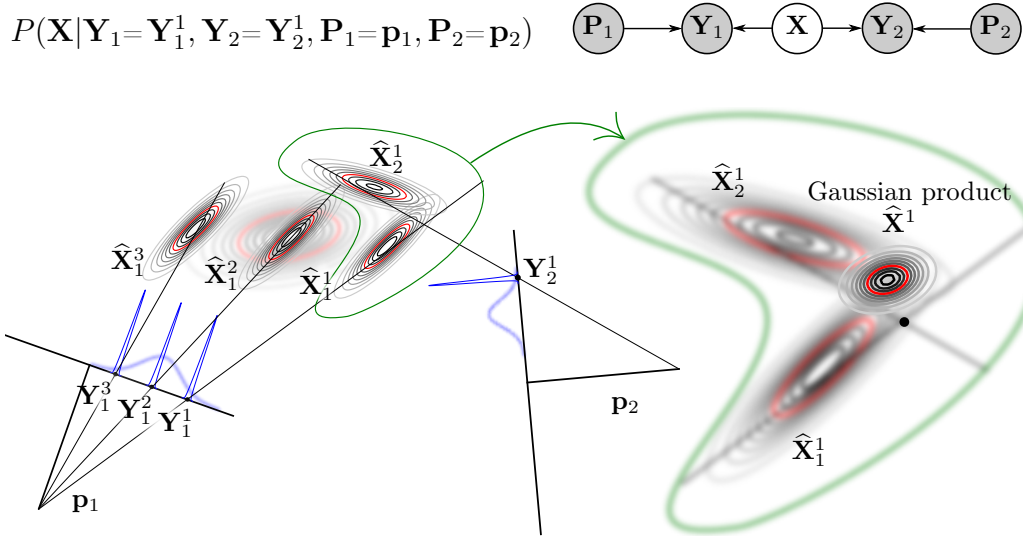


Figure 6.4: Back-projection and triangulation: given the joint Gaussians $P(\mathbf{X}, \mathbf{Y}_1 | \mathbf{P}_1 = \mathbf{p}_1)$ and $P(\mathbf{X}, \mathbf{Y}_2 | \mathbf{P}_2 = \mathbf{p}_2)$ from Figure 6.3, we obtain the estimated back-projections $\hat{\mathbf{X}}_n^i$ as $P(\mathbf{X} | \mathbf{Y}_n = \mathbf{Y}_n^i, \mathbf{P}_n = \mathbf{p}_n)$ by observing the projections \mathbf{Y}_n^i . Furthermore, we triangulate matching projections from different cameras by combining their back-projections to obtain $\hat{\mathbf{X}}^1$ as $P(\mathbf{X} | \mathbf{Y}_1 = \mathbf{Y}_1^1, \mathbf{Y}_2 = \mathbf{Y}_2^1, \mathbf{P}_1 = \mathbf{p}_1, \mathbf{P}_2 = \mathbf{p}_2)$. Note that the prior \mathbf{X} is not close to the posterior $\hat{\mathbf{X}}^1$ and, as a result, the estimation $\hat{\mathbf{X}}^1$ is at a slight offset from where the observed projections triangulate.

6.4.2 Full parameterisation example

Similar to the triangulation case, we can also parameterise the cameras with sigma points. We are then able to observe the projections in order to find information about the camera poses (as opposed to finding information about the structure of the scene in the case of triangulation). Figure 6.5 shows an example of a fixed point \mathbf{x} , and Gaussian camera parameters \mathbf{P}_1 and \mathbf{P}_2 parameterised as sigma points.

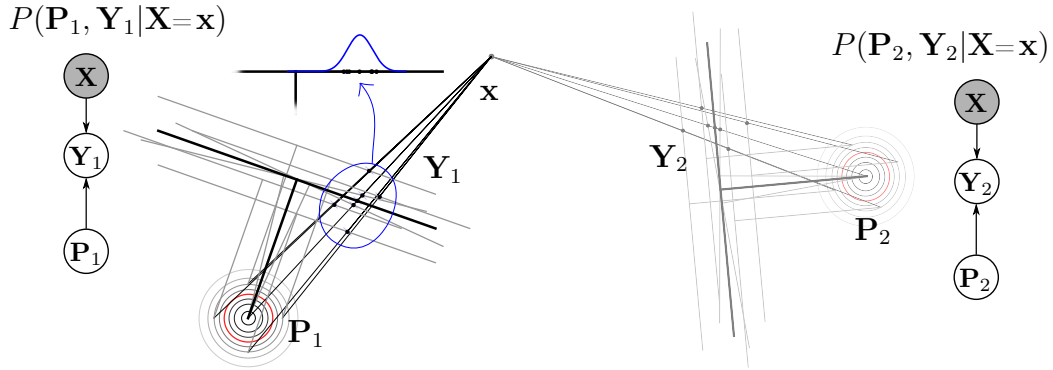


Figure 6.5: The Gaussian parameterisation of cameras: similar to Figure 6.3, but here \mathbf{P}_1 and \mathbf{P}_2 are parameterised as sigma points and a fixed point \mathbf{x} is projected using these sigma points as camera parameters, to obtain samples for the distributions over \mathbf{Y}_1 and \mathbf{Y}_2 .

Furthermore, we can combine the various Gaussian distributions in the system in order to obtain a joint Gaussian over all the variables. One straightforward approach is to take the combination of sigma points for more than one random vector as a new sigma point representation for the joint over those vectors. The size of this combination does not scale well for high-dimensional vectors so, as an alternative, a new covariance and mean can be built using the covariances and means of the original random vectors, which in turn can also be represented with sigma points. Note that no covariance between the different random vectors are captured by these methods. This approach is illustrated in Figure 6.6.

With these sigma point estimation methods we can investigate the relationships between the parameters of our structure and motion system. For a system with many parameters, we choose clusters within the system and pass messages between them to enable the determination of a posterior from the combined knowledge of all clusters.

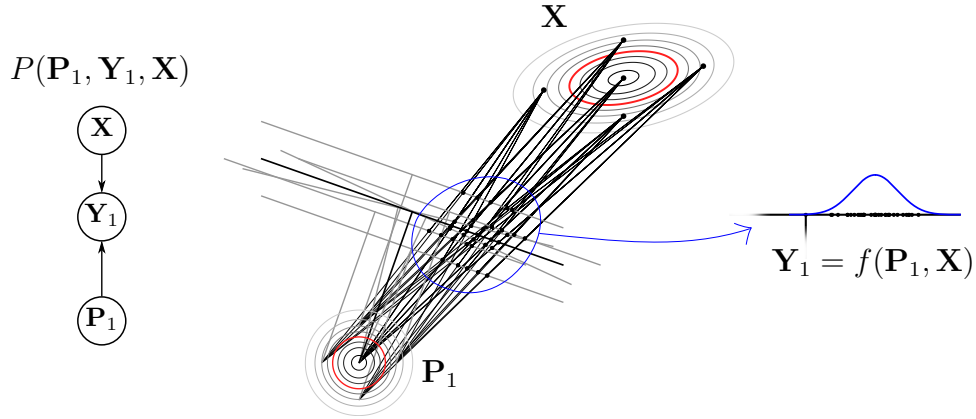


Figure 6.6: The Gaussian parameterisation of both points and cameras: we find the combination of sigma points for camera \mathbf{P}_1 and the point \mathbf{X} as new sigma points representing $P(\mathbf{X}, \mathbf{P}_1)$. Note that these composite sigma points capture no covariance between \mathbf{X} and \mathbf{P}_1 . We finally transform the composite sigma points to find a representation for \mathbf{Y}_1 , which leads to the joint Gaussian $P(\mathbf{X}, \mathbf{Y}_1, \mathbf{P}_1)$.

6.5 Building a cluster graph

We now move on to using the joint Gaussian distributions obtained through sigma points as clusters and building a cluster graph over which belief propagation can be performed.

So far we have a means of obtaining a joint Gaussian distribution for each projection $P(\mathbf{X}^i, \mathbf{Y}_n^i, \mathbf{P}_n)$, when given \mathbf{X}^i and \mathbf{P}_n as prior distributions. If all of these factors are used as clusters, the collection of clusters thus specified will contain all of the parameters of the system along with the projective relationships between them.

Since these clusters rely on prior parameters, we need to find an initial state for each of our parameters. One possibility is to use the basic multiple view geometry approach described in Chapter 3, and centre relatively wide Gaussian distributions around the result, as a means of obtaining a prior for the parameters of our system.

From the clusters a cluster graph is built using Du Preez's algorithm from Section 5.3. Figure 6.7 shows an example of a structure and motion cluster graph built from the example Bayes network in Figure 6.1.

Each observed variable, in our case each projected feature point, is initialised as a Gaussian distribution with mean taken as the measured value and standard deviation chosen to model the anticipated extent of measurement noise.

If all the information of the system is given and a cluster graph is connected, the final step is to apply belief propagation as described in Chapter 5 to obtain a posterior distribution of the structure and motion problem.

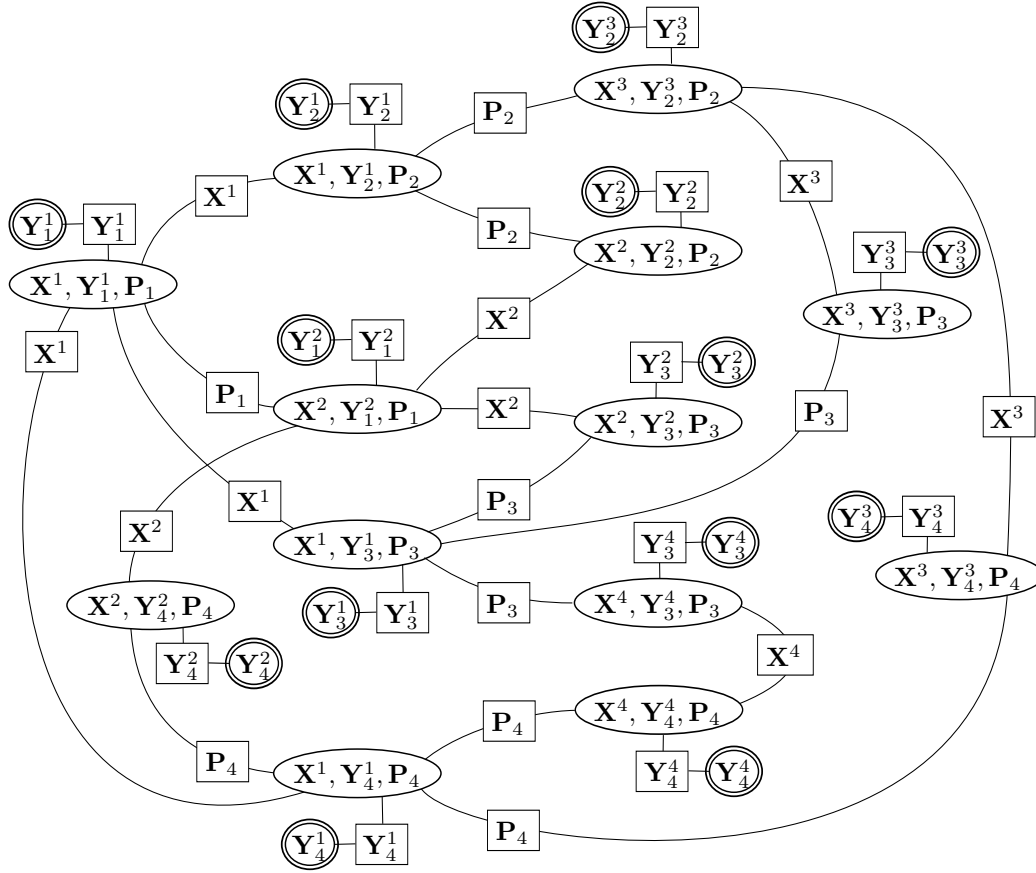


Figure 6.7: A cluster graph example containing all the random variables in the structure and motion problem depicted in Figure 6.1 and connected using Du Preez’s algorithm [46]. Each cluster represents a projection within the system estimated as a Gaussian random distribution using sigma points.

6.6 Finding the posterior distribution

We now illustrate how we solve a structure and motion problem using the probabilistic methods discussed. Furthermore, we show some difficulties with these methods and provide ideas on how to overcome them.

The effect of our approach is illustrated by means of a 2D structure and motion example. We generate a system with known ground truth as follows: we randomly position 2D points within certain bounds and place cameras randomly around these points. The 2D points are projected to the 1D image plane of every camera, and these projections (and the correspondences between them) serve as our observations. The task is to find a posterior distribution over the 2D points and camera parameters. As priors for this system, we choose a single position for all the 2D points and use the ground truth camera parameters with added noise, as can be seen in Figure 6.8 (a). We now have a system to solve.

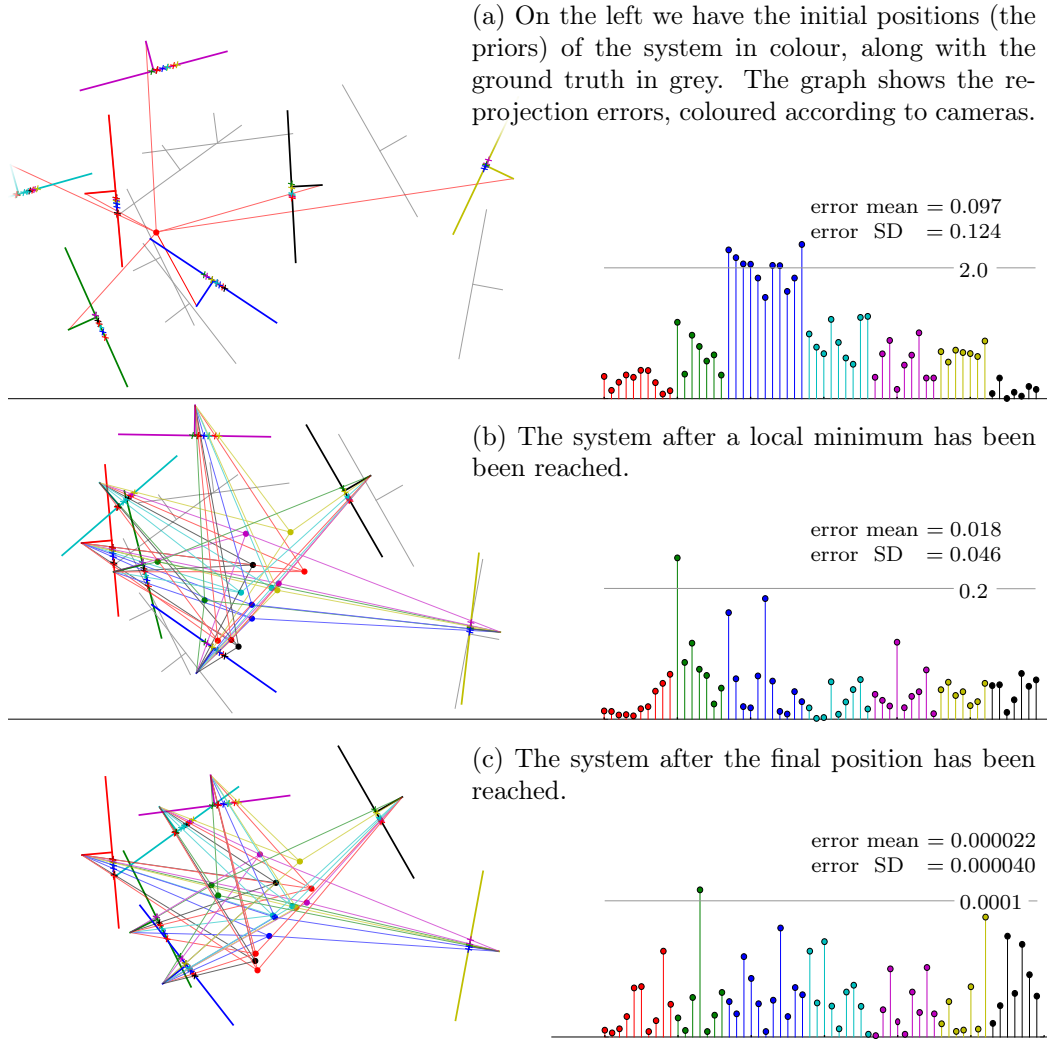


Figure 6.8: An example of solving a synthetic 2D structure and motion problem. Note that in order to avoid the visual effects of scale ambiguity, each plot on the left is translated and rotated according to the red camera and scaled according to the distance between the red and yellow camera.

We start by finding the clusters of the system and initialising them using the sigma point formulation. A cluster graph is then built using Du Preez's algorithm (similar to the one shown in Figure 6.7), on which we can run belief propagation. We observe the measurements by allowing each cluster to absorb the associated measurement. We do so by parameterising the measurement as a Gaussian with the mean as the measured value and the covariance as an estimation of the noise covariance. In our case we choose this covariance, which is a variance in our 1D case, as 10^{-7} (as a reference, the distance between the projection plane centre and the camera centre of each camera is 1).

Since the linear approximation of the system is derived from priors, the system models the space in the vicinity of these priors more accurately than the space further away. Therefore, if the priors are a poor representation of the ground

truth, the system will also poorly capture the projective geometry in the vicinity of the ground truth. This renders the posterior distribution untrustworthy. To circumvent this problem, we allow the system to recapture the projective geometry a posteriori, by repeating the following:

- reinitialise every cluster with the means and covariances of the current posterior distribution as new sigma point priors,
- run belief propagation on the new clusters to obtain yet a new posterior.

By following this approach, another complication arises: the covariances of the clusters are likely to get smaller for each new posterior distribution and, as a result, the system might converge to a local minimum that is still far from the ground truth. We remedy the situation by enlarging the covariances to allow our system to escape and explore the space outside such a local minimum. This is done with due consideration: if the covariances are haphazardly enlarged after every round of belief propagation, the linear estimation of the system will focus on too large a space and might not accurately capture the space around any of the minima. This might result in the system not converging at all, or converging to an unreliable solution.

Our approach is to allow the system to converge to a local minimum and only then enlarge the covariances to include a wider search space. We continue this process until the same minimum is reached twice in a row, by which point we take that as the final posterior. A further consideration would also be to reduce the amount by which the covariances are enlarged with each enlargement step. This allows the linear approximation to be more accurate as the system converges to a final position.

Refer to Figure 6.8 for a visual example of this approach. In conclusion, we managed to build a system which can probabilistically solve the 2D structure and motion problem. In the next chapter we extend this approach to 3D and evaluate the result.

Chapter 7

Experimental results

In this chapter we explore the limits of our structure and motion system. We first investigate how well the system operates with increasing amounts of noise in the parameters and later with how well the system operates when presented with measurements from real digital images.

To facilitate the first set of these tests, we generate synthetic data according to the following. We randomly generate 3D points in such a way that they are uniformly distributed within a sphere of radius 2. We also randomly generate cameras such that their centres are placed on a sphere of radius 10 and that they are facing the centre of the sphere, and we add Gaussian noise with a standard deviation of $\pi/20$ to the rotation parameters of the cameras and a standard deviation of $1/2$ to the positions. We find the input feature coordinates by projecting each 3D point onto a random choice of 2 to 6 cameras. An example of such a generated scene can be seen in Figure 7.1. This configuration is chosen to represent a typical structure and motion scenario.

The system we use for testing employs a pipeline similar to the one proposed in Chapter 6. We use a structure and motion estimate to build a prior for the system: we parameterise the 3D points and camera parameters as Gaussians, where each parameter is assigned some wide covariance and each parameter mean is taken as the originally estimated value. We then construct and initialise a cluster graph using Du Preez’s algorithm and sigma points.

We allow the system to repeatedly recapture the projective geometry, by repeating the following:

- reinitialise every cluster with the current posterior distribution as new sigma point priors,
- run belief propagation on the new clusters to obtain yet a new posterior.

For our tests we run the above-mentioned loop 60 times. We reset the cluster covariances every 3rd iteration (to allow for a wider search space) with increasingly narrower covariances (to allow finer convergence towards the end), with the exception of the last 6 iterations by which point we allow the system to

freely converge to the closest minimum. Also, we assign distributions for our measurements using an expected measurement noise parameter n to determine their covariance. We let the covariance grow exponentially smaller with each iteration starting with $(10)n I$ and ending with $(1/10)n I$. The approach is less dynamic than the one described in Chapter 6 in order to better compare between test results.

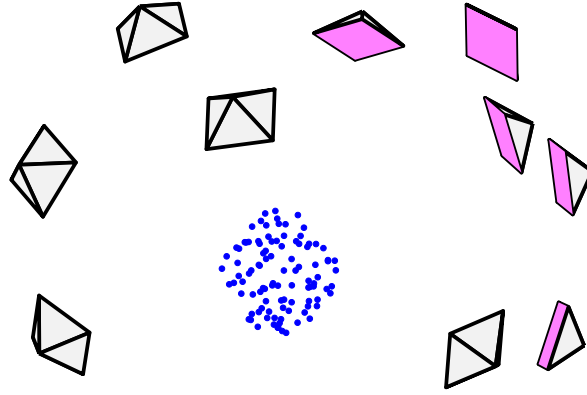


Figure 7.1: Synthetic data generated to test our system: the cameras and points are randomly generated to simulate a typical structure and motion problem.

7.1 Sensitivity to inaccurate priors

The aim of the first set of tests is to investigate how sensitive the system is to the accuracy of the prior model. For each test we generate a new scene (as described in the introduction of this chapter) to be used as the ground truth. As priors we place all the ground truth 3D points in the centre of our scene (so that the points are in front of all the cameras) and add Gaussian noise to the ground truth camera parameters. We then use our probabilistic graphical model approach to find a posterior distribution and test it against the ground truth. The setup, with the results documented in Figure 7.3, is as follows.

- What is kept constant between the test sets is the use of ground truth projections (without any added noise) as our observations.
- What varies between the test sets are the number of cameras, the camera poses, the number of 3D points, the coordinates of the 3D points, the intensity of the noise added to the camera poses and 3D points, and n (whose value is adjusted according to the difficulty of the test set).
- The aim is to test what effect the number of cameras, the number of points, and the intensity of the noise in the prior estimates of the camera poses and 3D points have on the accuracy of the posterior distribution and on the runtime of the system.

- We measure the accuracy of the posterior by using the average distance between the ground truth projections and the maximum likelihood estimates of the posterior projections, and we also measure the runtime of each test case.

			Standard deviation of Gaussian noise added						
Angles →			2.5°	5.0°	7.5°	10.0°	15.0°	20.0°	35.0°
Position →			0.5	1.0	1.5	2.0	3.0	4.0	7.0
C's ↓	P's ↓	n ↓	Prior error *, Posterior error *, Runtime *→*						
5	50	0.0001	0.1433 0.0002 56 s	0.2356 0.0005 75 s	0.2121 0.0005 72 s	0.3811 0.0004 78 s	0.7293 0.0033 134 s	3.1203 0.0020 131 s	1.7166 0.0113 289 s
5	100	0.0007	0.1495 0.0010 339 s	0.2174 0.0016 413 s	0.2289 0.0014 406 s	0.2366 0.0029 608 s	0.4150 0.0037 603 s	0.5091 0.0060 597 s	4.3998 0.0157 599 s
7	60	0.0004	0.1276 0.0004 60 s	0.1993 0.0010 62 s	0.3566 0.0019 101 s	0.2438 0.0026 101 s	0.3833 0.0030 88 s	0.7232 0.0036 144 s	8.6831 0.0217 223 s
10	100	0.0012	0.1419 0.0012 103 s	0.2156 0.0018 130 s	0.2976 0.0057 174 s	0.3697 0.0066 214 s	0.6045 0.0057 226 s	1.0230 0.0093 273 s	5.3663 0.0371 235 s
10	200	0.0020	0.1382 0.0014 774 s	0.2236 0.0031 896 s	0.2269 0.0034 1153 s	0.4711 0.0079 1059 s	0.9258 0.0147 1062 s	0.8315 0.0114 1120 s	4.0456 0.0335 1094 s
20	100	0.0016	0.1450 0.0014 46 s	0.2162 0.0029 59 s	0.2959 0.0033 66 s	0.4167 0.0067 71 s	1.2754 0.0058 65 s	0.9263 0.0139 71 s	7.3343 0.0321 80 s
20	200	0.0030	0.1471 0.0020 232 s	0.1998 0.0041 289 s	0.2989 0.0051 303 s	0.4000 0.0054 314 s	0.7239 0.0129 381 s	0.7868 0.0180 302 s	1.4711 0.0348 325 s
30	500	0.0040	0.1479 0.0030 1313 s	0.1972 0.0051 1368 s	0.2708 0.0088 1641 s	0.3212 0.0100 1411 s	0.6974 0.0163 1512 s	0.8402 0.0203 1539 s	2.8325 0.0530 1450 s

Figure 7.2: The test results regarding the effect of noisy priors on our system. The table presents the average projection error for the priors (in grey) and the posteriors (in black); and the runtime (from blue to red according to intensity) for different configurations of number of cameras (C's), number of points (P's), expected measurement noise n and amount of noise added to the parameters.

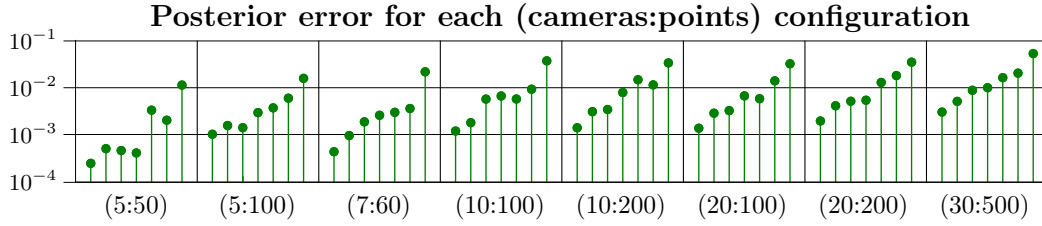


Figure 7.3: The average posterior errors in Figure 7.3 as a graph. The result of each camera and points configuration is graphed for each case of the 7 amounts of noise added.

The results show that the runtime of the system is very sensitive to both the size of the system (the number of points and cameras in a scene) and the level of noise inherent in the priors. The latter might be due to the fact that the belief propagation steps within the system generally take longer to reach a consensus when posed with conflicting information, such as noisy parameters. In fact, in the 30 cameras and 500 points configuration case the belief propagation steps did not obtain convergence and were prematurely terminated. In some of the other configurations the belief propagation steps also did not converge, such as in some of the 10 cameras and 200 points configuration cases.

The runtime regarding the 10 cameras and 200 points configuration seems to be in conflict with the much faster runtime of the 20 cameras and 200 points configuration. This is due to the size of the cluster graphs: since we have an additional cluster for each projection, each 3D point adds 2 to 6 clusters to the system, yet in contrast each additional camera adds no extra cluster to the system (in the case where 6 or more camera poses are already available). The underlying reason for the difference in runtime is the larger noise parameter n in the 20 camera and 200 points case, which makes the convergence criteria less strict.

The outcome also suggests that the accuracy of the posterior distribution is affected by the level of noise in the prior. This might be due to the case that when the prior is a large distance away from the global minimum, the system may have to move over a larger space to converge to that minimum. This allows more opportunity for the system to converge to a local minimum. Regarding the loss of accuracy with the increase of noise in the priors, it should be noted that in all of the 56 test cases the posterior model is still a more accurate representation of the ground truth than the prior model.

7.2 Sensitivity to measurement noise

The aim of the next set of tests is to investigate how sensitive our system is to measurement noise. In contrast to the previous set of tests, we generate a single scene consisting of 7 cameras and 60 points and generate a single prior distribution by adding Gaussian noise with a standard deviation of 5° to the

Euler angles of the cameras, adding zero mean Gaussian noise with a standard deviation of 1 to the camera centres, and placing all the 3D points in the centre of the scene. To test the effect of measurement noise, we let our system find multiple posterior distributions from this prior model by using observations with different levels of noise for each test case. The setup, with the results documented in Figure 7.4, is as follows.

- What is kept constant between the test sets are the number of cameras, the camera poses, the number of 3D points and the coordinates of the 3D points.
- What varies between the test sets are the amount of noise added to the measurements and n (the expected amount of noise) which we set equal to the amount of added noise.
- The aim is to test what effect the amount of measurement noise has on the accuracy of the posterior distribution and on the runtime of the system.
- We measure the accuracy of the posterior by using the average distance between the ground truth projections and the posterior maximum likelihood projections, and also between the prior projections and the posterior maximum likelihood projections. We also measure the runtime of each test case.

Measurement noise SD.	0.0005	0.001	0.005	0.01	0.015	0.02
-----------------------	--------	-------	-------	------	-------	------

In comparison with the ground truth

Prior error	0.23666	0.20047	0.17023	0.19410	0.19846	0.19673
Posterior error	0.00054	0.00097	0.00473	0.00935	0.01300	0.01811

In comparison with the noisy measurements

Prior error	0.23666	0.20049	0.16961	0.19414	0.19828	0.19748
Posterior error	0.00050	0.00098	0.00443	0.00845	0.01464	0.01924

Runtime	76s	78s	144s	197s	137s	224s
---------	-----	-----	------	------	------	------

Figure 7.4: The test results regarding the effects of noisy measurements on our system. The tables present the average projection error and runtime of 6 test cases, each with 7 cameras and 60 points, but with different levels of noise added to the measurement parameters.

For the range tested, it seems like the measurement noise has a linear influence on the average projection error in the posterior model. Since the measurements of our structure and motion model, i.e. the projections, are used as observations in our system and as the information relied upon to test the accuracy of our system, a correlation between the measurement noise and the posterior reprojection error is expected.

The results also suggest that the runtime of the system is negatively influenced by the amount of measurement noise in the model. Since we cannot assume that the level of measurement noise from a real-world dataset will be low, the slow runtime might limit the feasibility of the current system as a solution for real-world datasets.

7.3 Sensitivity to outliers

The aim of the next set of tests is to investigate how sensitive our system is to incorrect measurements. Similar to the previous test, we generate a single scene consisting of 7 cameras and 60 points and generate a single prior distribution by adding zero mean Gaussian noise of standard deviation 5° to the camera Euler angles, adding zero mean Gaussian noise of standard deviation 1 to the camera centres, and placing all the 3D points in the centre of the scene. To test the effect of incorrect measurements, we create a few test sets by allowing different numbers of tainted measurements. For a number m of tainted points in the system (with m an even number), we do the following $m/2$ times: randomly select one of the cameras and then swap the labels of two of its previously unswapped projections. We then find a posterior for this prior model. The test setup, with the results documented in Figure 7.5, is as follows.

- What is kept constant between the test sets are the number of cameras, the camera poses, the number of 3D points, the coordinates of the 3D points, the expected measurement noise parameter and the measurements (with no noise added).
- What varies between the test sets is the number of measurements tainted to be incorrect.
- The aim is to test what effect the number of incorrect feature matches (outliers) has on the accuracy of the posterior distribution and on the runtime of the system.
- We measure the accuracy of the posterior by using the average distance between all the points of the ground truth projections and the maximum likelihood estimates of the posterior projections, as well as between only the points of the ground truth projections and the maximum likelihood of the posterior projections that have not been tampered with. We also measure the runtime of each test case.

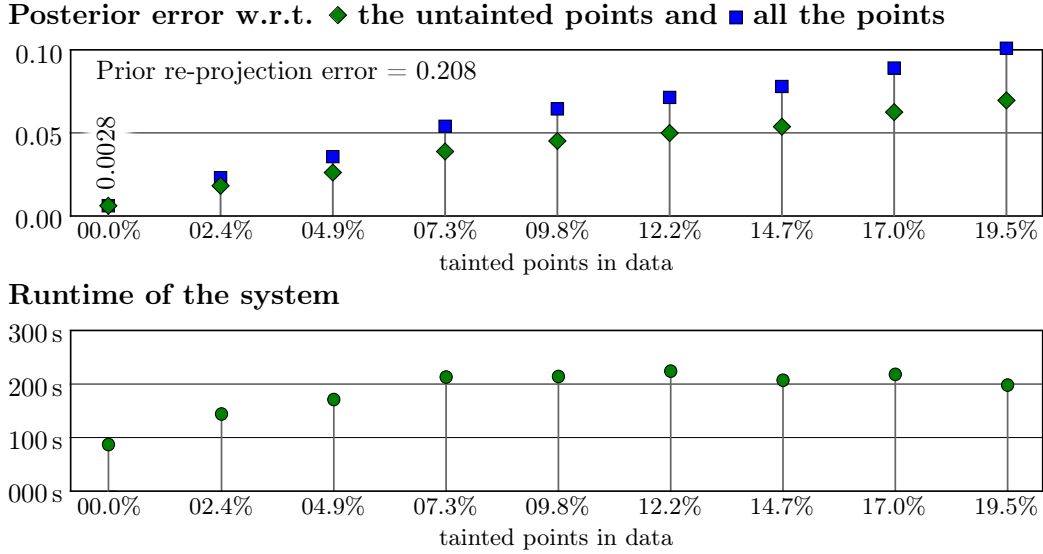


Figure 7.5: The test results with regards to the effect of incorrect feature matches (outliers) on our system. The prior model is corrupted by tainting the prior projection data with different percentages of incorrect measurements.

It seems that the reprojection error increases linearly with the number of tainted measurements in the system. It also seems like the rate by which the reprojection error increases for additional outliers is quite severe. For instance, the average reprojection error of the posterior compared to the ground truth increased from a value of 0.0028 for 0 tainted measurements to a value of 0.011 for 2.4% tainted measurements (which in this case is 6 out of 246 tainted projections). Although we expect real-world data to contain a small number of outliers, we do not have the means to assure that image correspondences are completely free from outliers.

The runtime of the system also does not scale well with an increase in outlier measurements. The runtime of the system doubles when about 7% of the measurements are tainted, after which the runtime seems to stabilise. It seems that the accuracy of the posterior model is affected more than the runtime of the system when introduced to outliers.

7.4 Real-world examples

For proof of concept we also applied our system to images taken of real scenes. By applying the feature extraction and multiview geometry theory from Chapter 2 and Chapter 3 on a set of images, we obtained camera positions and 3D points that could be used as priors for our probabilistic structure and motion system.

We tested the pipeline on two datasets, with the following results. For the first image set (Figure 7.6), we obtained 11 cameras and 745 3D points, which we reduced to 198 points by the constraint that points have to be visible to at least three cameras. Our system managed to decrease the average projection error from 12.51 pixels in the prior model to 1.34 pixels in the posterior model. For the second image set (Figure 7.7), we extracted 26 cameras and 996 points and reduced the average projection error from 3.3 pixels to 2.2 pixels using our system. In this case the system did not converge to a solution within a reasonable time frame and was terminated before a local minimum was reached. Although the posterior model is not optimal, it is still interesting that the projection error improved.

The measurements in the real-world data are likely quite noisy and we do not have certainty that all the incorrect feature matches have been removed effectively by RANSAC. From previous tests we have established that both of these problems have a negative effect on the runtime of our system and the chances for converging within a reasonable time frame. There is clear promise in the performance of our system, especially on small datasets, but it has to be acknowledged that some improvements are needed before it can be used as a practical, robust and general solution for a wider range of real-world structure and motion problems.

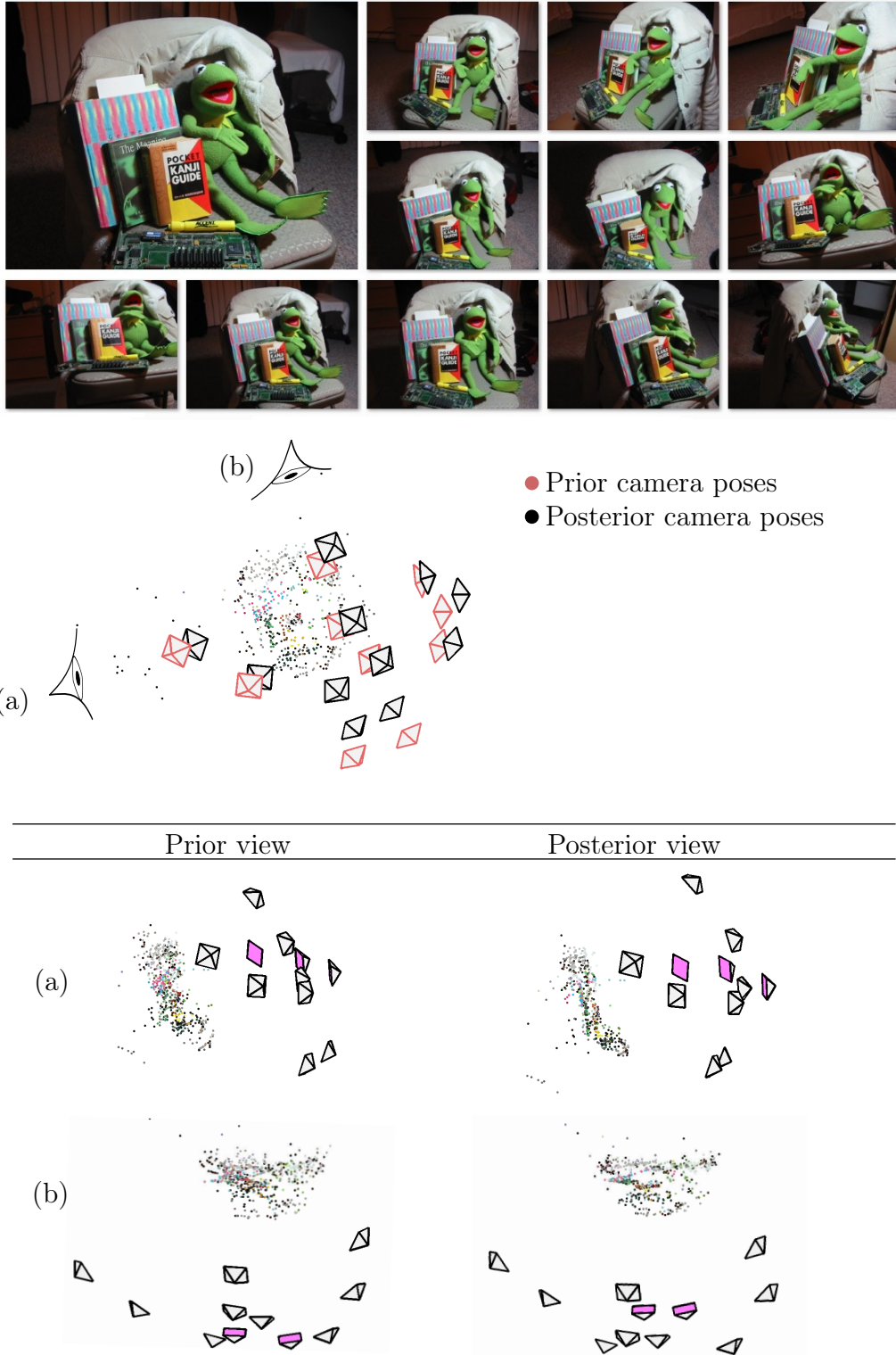


Figure 7.6: A real-world structure and motion example using our proposed pipeline. Note the two viewpoints (a) and (b): the posterior view seems to fit the surfaces more accurately than the prior view. The triangulated points representing the books are less disperse and a closer representation to flat surfaces in the posterior view than in the prior view.

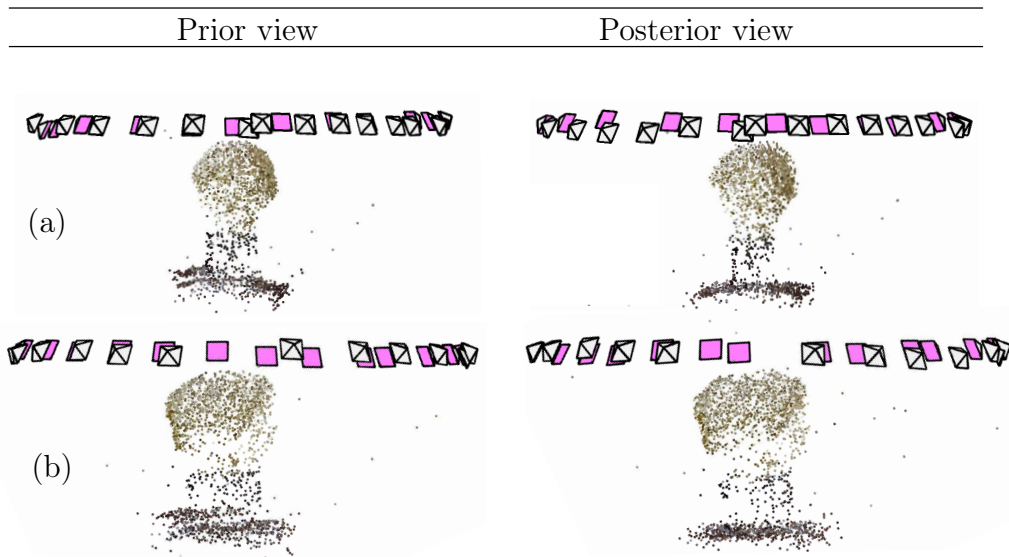
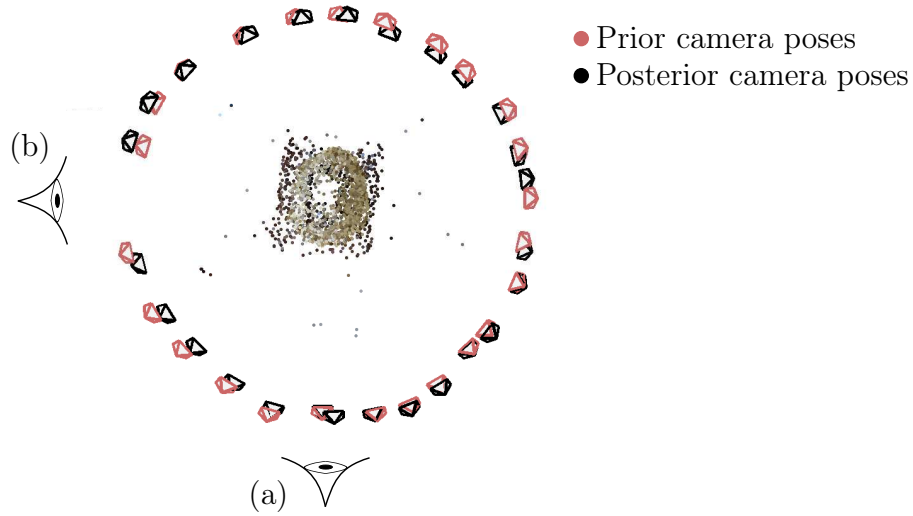


Figure 7.7: A real-world structure and motion example using our proposed pipeline. This example did not converge within a reasonable time frame, and therefore a local minimum was not reached. Note the two viewpoints (a) and (b): the posterior does exhibit interesting attributes, e.g. the table's surface appears to be more regular.

Chapter 8

Conclusions and future work

The main objective of this study was to investigate the extent to which a probabilistic graphical model approach can solve a nonlinear problem. Although this study specifically focused on the structure and motion problem, our secondary objective was to keep the implementation as general as possible in order to be applicable to other nonlinear problems.

We discussed the typical feature based structure and motion pipeline and the underlying multiview geometry theory. Furthermore, we discussed probabilistic graphical models, along with the tools and theory necessary to construct and solve one. We were able to successfully combine the theory from these two somewhat separate fields and provide a probabilistic graphical model solution to the structure and motion problem.

Although the focus of graphical models is usually on systems with a discrete representation, or systems with linear dependencies, one of the main contributions of this study is that we provide a system that can handle nonlinear relationships between variables. This is accomplished by capturing all the relationships between random variables within a system as covariances, by means of sigma point parameterisation.

We initially attempted to use this approach directly, but determined that the sigma point parameterisation of the priors does not always provide an accurate linear estimation of the problem, especially if the prior beliefs are far from the ground truth. As a result such a system converges to a suboptimal space. We were able to circumvent this problem by allowing the system to converge, but then reparameterising everything with sigma points (using the current posterior beliefs), and repeating this process until the system converges to the same space twice.

We further discussed the construction and parameterisation of our system by means of implementing a 2D structure and motion problem. The approach was quite general, the system behaved quite well, and was able to efficiently converge to a space equivalent to the ground truth. This was achieved by providing nothing more than

- imprecise estimates for the 2D points and cameras as prior beliefs,
- the projected points as observations, and
- the forward projection function f_{2D} as the relationship between variables.

Furthermore, we tested our system on the 3D structure and motion problem by initialising the system in a fashion similar to the 2D case. The pipeline discussed in Chapter 3 was used to obtain an estimate for our prior model. We made the following observations:

- the system is able to successfully resolve the structure and motion problem and can converge to a space equivalent to the ground truth,
- the runtime of the system needs some overall improvement,
- the runtime of the system is very sensitive to the level of noise in the priors, and to the size of the cluster graph used,
- as expected, the level of noise in the measurements has a somewhat linear correlation with the level of noise in the posterior, and
- the number of outliers in the measurements has a somewhat linear effect on the runtime and accuracy of the system, but the accuracy rate of decline is quite steep.

Finally, the approach was tested on two real-world datasets. The one dataset with 11 cameras and 745 3D points behaved quite well and the system managed to get a projection error reduction from 12.51 pixels to 1.34 pixels. For the other dataset with 26 cameras and 996 3D points the system was unable to reach convergence. This clearly shows that our system can perform well on small real-world datasets, but may not yet be suitable for larger ones.

In conclusion, the system is able to reach its design goals and solve the structure and motion problem by producing useful results. We have to acknowledge though, that the system requires some improvements before it can be seen as a robust solution for a wide range of real-world structure and motion problems. Finally, we were able to provide a general system that can be used on a wide range of nonlinear problems that require the modelling of uncertainty and statistical dependencies.

8.1 Future work

The system developed in this thesis serves as a proof of concept that probabilistic graphical models can be used as a solution to a nonlinear problem, and the results were found to be promising. However, we now discuss the many areas that can still be improved upon and suggest possible avenues of further research.

For one, the runtime and convergence of our system is possibly the biggest limiting factor. Koller [44] suggests that a lot of severely conflicting information has a drastic influence on the convergence of a probabilistic graphical model. This might be one of the reasons we are experiencing poor runtime. If this is true, it might be due to the fact that the linearisation of interrelationships from sigma points can be quite inaccurate, especially when the prior beliefs of a system are far from the ground truth. Currently our system performs full belief propagation multiple times, each time on slightly more accurate prior beliefs. This is far from ideal and, therefore, we suggest to investigate the following idea. Rather continuously relinearise the cluster parameters with sigma points during belief propagation. This might possibly be expensive, but has the potential to eliminate the need for multiple rounds of full belief propagation. Since the linearisation would follow the current beliefs of the system, this might lead to faster and more robust convergence.

We may also consider incorporating additional logic into our system, such as penalising conflicting parameters, or integrating the intrinsic camera parameters as additional random variables in the system.

The only linearisation technique used throughout this study is the sigma point parameterisation. It might be beneficial to investigate other types of linearisation techniques to be used within the system. Furthermore we might want to investigate the performance of tailor-made solutions compared to our sigma points parameterisation approach.

Since our system is quite general, we suggest an investigation into other non-linear problems that might be solvable with this system. Possible suggestions include structural design, thermodynamics, or any system with a large number of random variable transformations.

List of References

- [1] Autodesk 123D catch: Generate 3D model from photos. [Online]. Available: <http://www.123dapp.com/catch>
- [2] 123D catch - create your own 3D model: Planning your shoot. [Online]. Available: www.youtube.com/watch?v=7TfXXJxDsXw
- [3] D. Crandall and N. Snavely, “Modeling people and places with internet photo collections,” *Communications of the ACM*, vol. 55, no. 6, pp. 52–60, 2012.
- [4] Photo tourism: exploring photo collections in 3D. [Online]. Available: <https://http://phototour.cs.washington.edu/>
- [5] A. Kushal, B. Self, Y. Furukawa, D. Gallup, C. Hernandez, B. Curless, and S. Seitz, “Photo tours,” in *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIM-PVT)*. IEEE, 2012, pp. 57–64.
- [6] The Flyover feature of Apple Maps. [Online]. Available: <https://mapsconnect.apple.com/>
- [7] 3DF Zephyr Pro: 3d models from photos. [Online]. Available: <http://www.3dflow.net/>
- [8] Neitra 3D Pro: detailed 3D modeling and reconstruction of real objects from images. [Online]. Available: <http://triayaam.com/>
- [9] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [10] D. Lowe, “Object recognition from local scale-invariant features,” in *1999 IEEE International Conference on Computer Vision*, vol. 2. IEEE, 1999, pp. 1150–1157.
- [11] D. Nister and H. Stewenius, “Scalable recognition with a vocabulary tree,” in *2006 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2. IEEE, 2006, pp. 2161–2168.
- [12] D. Crandall, A. Owens, N. Snavely, and D. Huttenlocher, “Discrete-continuous optimization for large-scale structure from motion,” in *2011 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2011, pp. 3001–3008.

- [13] M. Fischler and R. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [14] M. Lourakis and A. Argyros, “The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm,” Institute of Computer Science, Foundation for Research and Technology, Hellas, Tech. Rep., 2004.
- [15] N. Snavely, S. Seitz, and R. Szeliski, “Photo tourism: exploring photo collections in 3D,” in *ACM transactions on graphics (TOG)*, vol. 25, no. 3. ACM, 2006, pp. 835–846.
- [16] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. Seitz, and R. Szeliski, “Building Rome in a day,” *Communications of the ACM*, vol. 54, no. 10, pp. 105–112, 2011.
- [17] J. Kopf, M. Cohen, and R. Szeliski, “First-person hyperlapse videos,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 78, 2014.
- [18] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, “Bundle adjustment—a modern synthesis,” in *Vision algorithms: theory and practice*. Springer, 2000, pp. 298–372.
- [19] S. Andreassen, F. Jensen, and K. Olesen, “Medical expert systems based on causal probabilistic networks,” *International Journal of Biomedical Computing*, vol. 28, no. 1, pp. 1–30, 1991.
- [20] M. Malfait and D. Roose, “Wavelet-based image denoising using a Markov random field a priori model,” *IEEE Transactions on Image Processing*, vol. 6, no. 4, pp. 549–565, 1997.
- [21] Z. Tu and S. Zhu, “Image segmentation by data-driven Markov chain monte carlo,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 657–673, 2002.
- [22] P. Anantharam, K. Thirunarayan, and A. Sheth, “Traffic analytics using probabilistic graphical models enhanced with knowledge bases,” *Proceedings of the 2nd International Workshop on Analytics for Cyber-Physical Systems (ACS-2013)*, 2013.
- [23] S. Julier and J. Uhlmann, “New extension of the Kalman filter to nonlinear systems,” in *AeroSense 97 Conference on Photonic Quantum Computing*. International Society for Optics and Photonics, 1997, pp. 182–193.
- [24] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [25] M. Muja and D. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *2009 International Conference on Computer Vision Theory and Applications*, vol. 2, 2009.

- [26] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *2006 European Conference on Computer Vision*. Springer, 2006, pp. 404–417.
- [27] M. Agrawal, K. Konolige, and M. Blas, "CenSurE: Center surround extremas for realtime feature detection and matching," in *2008 European Conference on Computer Vision*. Springer, 2008, pp. 102–115.
- [28] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, 2010.
- [29] E. Mair, G. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," in *2010 European Conference on Computer Vision*. Springer, 2010, pp. 183–196.
- [30] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," in *2010 European Conference on Computer Vision*. Springer, 2010, pp. 778–792.
- [31] S. Leutenegger, M. Chli, and R. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *2011 IEEE International Conference on Computer Vision*. IEEE, 2011, pp. 2548–2555.
- [32] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *2011 IEEE International Conference on Computer Vision*. IEEE, 2011, pp. 2564–2571.
- [33] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast retina keypoint," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 510–517.
- [34] H. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover." Stanford University, Department of Computer Science, Tech. Rep., 1980.
- [35] C. Harris and M. Stephens, "A combined corner and edge detector." in *Proceedings of the Fourth Alvey Vision Conference*, vol. 15, 1988, p. 50.
- [36] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, "Evaluation of low-complexity visual feature detectors and descriptors," in *2013 IEEE International Conference on Digital Signal Processing*. IEEE, 2013, pp. 1–7.
- [37] D. Capel, "An effective bail-out test for RANSAC consensus scoring." in *2005 British Machine Vision Conference*, 2005.
- [38] O. Chum and J. Matas, "Matching with PROSAC - progressive sample consensus," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 220–226 vol. 1.

- [39] T. Sattler, B. Leibe, and L. Kobbelt, "SCRAMSAC: Improving RANSAC's efficiency with a spatial consistency filter," in *2009 IEEE International Conference on Computer Vision*. IEEE, 2009, pp. 2090–2097.
- [40] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.
- [41] G. Bradski, "The OpenCV library," *Dr. Dobbs's Journal of Software Tools*, 2000.
- [42] J. Bouguet, "Camera calibration toolbox for matlab." [Online]. Available: www.vision.caltech.edu/bouguetj/calib_doc
- [43] M. Lhuillier and L. Quan, "Surface reconstruction by integrating 3D and 2D data of multiple views," in *2003 IEEE International Conference on Computer Vision*. IEEE, 2003, pp. 1313–1320.
- [44] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- [45] D. Tylavsky and G. Sohie, "Generalization of the matrix inversion lemma," *Proceedings of the IEEE*, vol. 74, no. 7, pp. 1050–1052, 1986.
- [46] J. du Preez, private communication, Stellenbosch University, 2014–2015.
- [47] E. Wan and R. van der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000*. IEEE, 2000, pp. 153–158.
- [48] J. du Plessis, "Performance evaluation of EMDWs cluster graph algorithms," Bachelor Thesis, Stellenbosch University, September 2015.
- [49] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *The algorithms of Kruskal and Prim*. MIT press, 2009.
- [50] M. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, 1987.
- [51] C. Bishop, *Pattern recognition and machine learning*. Springer New York, 2006.
- [52] A. Ihler, J. Iii, and A. Willsky, "Loopy belief propagation: Convergence and effects of message errors," in *Journal of Machine Learning Research*, 2005, pp. 905–936.
- [53] P. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences, Calcutta*, vol. 2, pp. 49–55, 1936.